CEREBRA: A 3-D VISUALIZATION AND PROCESSING TOOL FOR BRAIN NETWORK EXTRACTED FROM FMRI DATA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BARIŞ NASIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2017

Approval of the thesis:

CEREBRA: A 3-D VISUALIZATION AND PROCESSING TOOL FOR BRAIN NETWORK EXTRACTED FROM FMRI DATA

submitted by **BARIŞ NASIR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Fatoş T. Yarman Vural
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

Prof. Dr. Fatoş T. Yarman Vural
Computer Engineering Department, METU _____

Prof. Dr. Nafiz Arıca
Computer Engineering Department, Bahçeşehir University _____

Assoc. Prof. Dr. Ahmet Oğuz Akyüz
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering Department, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: BARIŞ NASIR

Signature :

ABSTRACT

CEREBRA: A 3-D VISUALIZATION AND PROCESSING TOOL FOR BRAIN NETWORK EXTRACTED FROM FMRI DATA

Nasır, Barış
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Fatoş T. Yarman Vural

August 2017, 123 pages

In this thesis, we introduce a new tool, CEREBRA, for visualizing 3D network of human brain, extracted from the functional magnetic resonance imaging (fMRI) data. The tool aims to visualize the selected voxels as the nodes of the network and the edge weights are estimated by modeling the relationships among the voxel time series as a set of linear regression equations. This way, researchers can analyze the active brain regions/voxels and observe the interactions among them by analyzing the edge weights and node degree distributions of the brain network, for the underlying brain state(s). CEREBRA provides an easy to use interactive interface with basic display options for users to examine the details of the brain network. CEREBRA simplifies the network by built-in processors of graph reduction algorithms to display various properties of the network. The reduction algorithms vary from basic filtering methods to more complex graph sparsifier metrics. The toolbox is, also, capable of space-time representation of the dynamically changing voxel intensity and edge strength values, by animating the 3D voxel time series.

Keywords: Visualization, Graph Sparsification, Brain Graph, fMRI, 3D

ÖZ

CEREBRA: FMRG VERİSİNDEN ELDE EDİLEN BEYİN AĞLARINI 3-B GÖRSELLEME VE İŞLEME ARACI

Nasır, Barış
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Fatoş T. Yarman Vural

Ağustos 2017, 123 sayfa

Bu çalışmada, fonksiyonel manyetik rezonans görüntüleme (fMRG) verisinden elde edilen 3 boyutlu insan beyni çizgesini görselleyebilen yeni bir yazılımı, CEREBRA'yı sunuyoruz. CEREBRA, küçük hacim birimlerini (voksel), ağın düğümleri olarak görsellerken; bu düğümler arasındaki kenarlar da düğümlerin zaman içindeki ilişkilerinin bir dizi doğrusal regresyon yöntemleri ile yakınsanması olarak modellenmiştir. Bu sayede araştırmacılar beyin üzerinde daha aktif çalışan bölgeleri ve bölgeler arasındaki ilişkiyi, kenar ağırlıklarına ve düğüm derece dağılımına bakarak inceleyebileceklerdir. Geliştirilen araç, basit bir arayüze sahip olmakla birlikte beyindeki detayları daha kolay inceleyebilmek amacıyla basit görüntüleme seçenekleri sunmaktadır. Sahip olduğu çizge sadeleştirme algoritmaları ile beyin ağının farklı özelliklerini sade bir şekilde gösterebilmektedir. Çizge sadeleştirme algoritmaları basit filtreleme methodlarından, daha karmaşık çizge sadeleştirme ölçütlerine kadar geniş bir yelpazeye yayılmaktadır. Ayrıca, geliştirilen araç, beyin ağını voksellerin ve kenarların zaman

içindeki değişimlerini renk kodu olarak değiştirebilmektedir. Bu da fMRG deneyi sırasında beyinde gerçekleşen bilişsel süreçleri gözlemleme imkanı sunmaktadır.

Anahtar Kelimeler: Görselleme, Çizge Sadeleştirme, Beyin Ağı, fMRG, 3-B

To my family...

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Prof. Dr. Fatoş Tünay Yarman Vural for accepting me to her work group and encouragement throughout my M.Sc. years. Besides from being an exceptional academician; her personal values, compassion and graceful attitude make her a very special one. She has been a great inspiration for me as a person and an academician. It has been a privilege to work with her.

I would like to thank my family for all their support and patience. I am very grateful to my father for relieving the pressure on my shoulders, my dearest mother who is able to make me smile anytime and of course my sister for her support with those fancy snapchat filters.

I also wish to thank my lovely Zeynep for her endless patience, support and care. During this study, I once again realized that how lucky I am.

My special thanks go to my colleagues at Image Laboratory. Especially, to Burak Velioğlu, Emre Aksan and Ozan Yıldız for helping me to test the proposed work and for their comments. Then, to Arman Afrasiyabi and Hazal Moğultay for using this tool on their works. It was an honor to work in this laboratory.

I could not pass Tuncay Korkmaz and Bitli Notalar Kumpanyası without thanking them for letting me breathe during this intense and tedious thesis period. It is a full of joy and pleasure playing with those precious people.

Last but not the least, I would like to thank Bahattin Tozyılmaz, Atakan Kaya, and Özlem Ceren Şahin for their efforts in the first version of this work in our senior year.

I acknowledge the support of TÜBİTAK (The Scientific and Technological Research Council of Turkey) within the scope of project 114E045 during my M.Sc. research.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

fMRI	Functional Magnetic Resonance Imaging
BOLD	Blood Oxygenation Level Dependent
3D	Three Dimensional
MNI	Montreal Neurological Institute
AAL	Automated Anatomical Labeling
OpenGL	Open Graphics Library
OPENGLSL	Open Graphics Library Shader Language
N-Cut	Normalized Cut
LMM-TM	Local Mesh Model with Temporal Measurements
HRF	Hemodynamic Response Function
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ROI	Region of Interest

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Visualization is the field of transforming any data recorded under physical phenomenon to visual representations for effectively analyzing the underlying process that make it possible to gain insight from data [59]. The most common visual representations such as line graphs, from millennium years ago, bar charts, pie charts, circle graphs and histograms are used in our everyday life by a wide range of professionals ranging from scientists to bank officers. These techniques help us to support data analysis for decades.

However, today, visualization is much more than a collection of plots and charts. There are specific visualization techniques for many data forms, including, but not limited to, texts [76, 38, 26], networks [27, 10, 44, 64], image collections and videos [47, 16], mathematical models (geometrical, statistical, etc.) [28], historical or progressive records [36], dynamic data-driven models [18], algorithms and data structures [29, 45], time-series [30, 74], multivariate data [24, 72] and a wide range of domain-specific data in disciplines such as economics [35], biology and medicine [68], which is the target area of this work.

In this thesis, we aim to develop a visualization tool for the human brain, which is inherently a very large scale dynamic network with billions of massively interacting neurons. Recent advances in functional magnetic resonance imaging (fMRI) techniques enable us to observe some of the network properties of the human brain, in-

directly, by measuring the blood oxygenation level dependent (BOLD) time series at each volume element on a three-dimensional discrete grid [23, 48]. Loosely saying, the BOLD signal increases as the neurons in a volume element needs more oxygen. This volume element, called voxel, may include a couple of thousands of neurons, depending on the characteristics of the fMRI machine. In other words, a brain graph formed by voxels is a representation of the actual brain in very low resolution. A typical fMRI recording generates a brain volume of size $100{,}000$ to $200{,}000$ voxels, at approximately each 2 seconds [40]. Therefore, for a period of 10 minutes, fMRI machine generates 300 to 600 brain volumes. In other words, at each voxel, a time series of length 300 to 600 samples are recorded. Since the recorded data is a very curse and indirect representation of the brain network, considering the number of voxels and the corresponding adjacency matrix, the researchers face with a serious visual representation problem.

A typical fMRI data with $100{,}000$ voxels may have an adjacency matrix up to 10^{10} elements. Visualizing this number of edges in a rigid volume like brain creates an inevitable "hairball" problem. In a word, the displayed 3D graph will have many intersecting edges that block the voxels and even most of the other edges. One solution to avoid this problem is to display only a part of the graph, depending on the specific goal of the neuroscientific study. For example, one may be interested only one representative voxel from each anatomical region. However, this solution makes the analysis of relations within a region impossible. In addition, it removes the cluster information embedded in the unprocessed data. By clusters, we mean that the functional groups formed by voxels, which are generally detected by the functional relations between voxels. As these relations are indicated by edges, collapsing all of them according to anatomical information may lead information loss.

Another solution to avoid hairball problem is to provide a simple thresholding methods to remove edges whose weights lie between certain values. This allows users to observe the strongest positively or negatively correlated voxels or both at the same time (by blocking only the intermediate values). Although this solution may work on some situations such as, observing the most correlated voxel or region groups,

it may remove all the connections in small clusters which result in non-negligible information loss.

The visualization technique may display only a selected number of regions (two or three clusters or anatomical region) according to the cognitive task(s) to be studied at a time to get rid of hairball problem. While this solution makes the display much more clear, the users may not be able to observe inter-relations between all regions at once. As a result, it may be difficult for the user to observe hub regions and their connections.

In this thesis, we have proposed a tool that combines multiple visualization techniques into one package to provide a better display to the user with respect to his/her goals. The user may combine several functionalities to get the desirable graph view.

1.2 Proposed Work and Contributions

In this thesis, the proposed visualization tool for brain network, called CEREBRA, provides additional features on the existing solutions, which transform the complex graph structure into a more comprehensible and simple one. CEREBRA represents the brain data in a 3D space as a function of time. It enables researchers to analyze anatomical, functional and structural information which can be extracted from the fMRI data. The set of voxels represent the nodes of the brain network in three-dimensional Euclidean space and the estimated relations between the voxel pairs are represented by weighted edges. Each node, corresponds a voxel in three-dimensional brain volume, has coordinates (x, y, z) together with a time series recorded under a predefined cognitive state. CEREBRA is capable to represent the brain network in different scale orientation and translation, allowing zooming in and out in a specific brain region. CEREBRA provides some additional features onto two commonly used brain visualization tools available in the literature [77, 25], which are listed below:

- We have developed a tool that visualizes the brain network with the time series information to observe the voxel intensity changes as a function of time. The

fMRI data recorded under a cognitive experiment is illustrated by CEREBRA as an animation, in which the voxel and edge colors change according to their intensity/weight fluctuations in time. This feature enables researchers to track the brain activity recorded during the time course of the fMRI experiment.

- CEREBRA provides a set of filters to clean up "undesirable" voxels and edges in an estimated brain graph. The filters are implemented to bypass brain connectivity hairball problem. They remove the edges according to connection weight and local thresholding methods.

- The proposed tool is capable of displaying multivariate brain data by changing voxel colors according to the selected modality. For example, voxel colors can be arranged according to activity records. Also, the proposed tool is capable of editing the brain graph by evaluating in-degree, weighted in-degree, out-degree, weighted out-degree, total degree and weighted total degree values of the graph. The tool is capable of coloring the voxels according to anatomic regions or brain clusters.

- We have enabled users to load their machine learning algorithm results to analyze formed groups on the brain graph and/or connectivities among them for the underlying cognitive state(s). If any functional region information is available and loaded, voxel color codes are arranged according to the region labels. Each label is represented by a unique color code. In addition, the user may limit the number of displayed regions using the loaded region information and view other modalities on the selected regions.

- In CEREBRA, anatomical region information is pre-loaded into the system for MNI (Montreal Neurological Institute) coordinate space. Similar to the functional regions, voxels could be colored according to their anatomical region information where each region is represented by a unique color or only selected regions could stay on the display with another modality color map.

- The available brain visualization tools, such as BrainNet Viewer [77] and Marsbar [13] are implemented in MATLAB and only employ .mat files. On the other hand, CEREBRA is implemented in C++ to beat performance issues related

with MATLAB. Furthermore, CEREBRA is capable of reading and processing .mat files. In other words, the tool combines the traditional approaches with the power of C++ and OpenGL (and OpenGLSL in order to benefit from GPU) simultaneously.

- Although the toolbox serves sophisticated sparsification and simplification algorithms, it offers a user-friendly graphical interface. A special effort is spent to design simple panels with well-defined buttons on the screen so that the users are not lost in multiple options and parameter spaces.

1.3 Organization of the Thesis

In Chapter 2, a literature survey on general purpose graph visualization tools and brain specific visualization tools are overviewed. Moreover, the nature of the fMRI data and methods to pre-process it, including estimating the connections between neuronal units, are summarized. Then, the graph structure that holds the data is introduced. The section ends with explaining one of the most crucial parts of this study, reducing the number of edges on the 3D graphs.

Chapter 3 provides detailed information about the input structure and processing of CEREBRA. In this chapter, supported file formats and their structures are introduced. The section continues with explaining the class structure in CERREBRA that holds the data itself. We conclude the chapter by explaining the post-processing steps on the loaded data.

Chapter 4 contains the design and implementation structure of CEREBRA. The section starts with giving an insight about how we design CEREBRA as a collection of sub-systems and what functionalities are embedded into each sub-system is explained. Then, the section provides the coloring scheme for voxels and edges with possible options and examples. In addition, the details about animating time-series information are also given in this chapter. The chapter continues with explaining the display options such as, changing voxel/edge sizes and normalizing values between the user defined minimum and maximum values to get a standard color scheme. Then,

a simple algorithm for computing Pearson correlation coefficient among voxel time series is detailed with its implementation and example figures. After explaining the coloring steps and construction of the connections, the chapter continues with focusing different intensity/weight intervals on the brain graph using basic thresholding. Moreover, a more advanced method, local graph sparsification is also explained in Chapter 4. We have explained how we extend the existing method proposed by Satuluri et al. [63] and its usage on different scenarios. The chapter ends with explaining the affine transformations that could be operated on the graph interactively.

In Chapter 5, the final chapter, we share the development environment details, discuss the study and future plans for the project.

CHAPTER 2

BACKGROUND FOR GRAPH VISUALIZATION AND FUNCTIONAL MAGNETIC RESONANCE IMAGING DATA

In this chapter, a literature survey about the general purpose graph visualization tools and the methodologies used in this thesis are presented with the intention of providing the reader a background. Firstly, the popular graph visualization tools such as Pajek and JUNG will be overviewed. Then, brain specific visualization tools, namely, BrainNet Viewer and Connectome Viewer Toolkit, are discussed considering their pros and cons. The section continues with giving some insights into functional magnetic resonance imaging (fMRI) data and preprocessing operations applied on row fMRI data to give details about the structure of the input and anatomical mappings of fMRI. Then, we have introduced the methods to estimate the connections between voxel pairs in the human connectome. After describing the input itself, we introduce graph structure representation of fMRI data. The section ends by detailing the methods to reduce the number of connections in an input graph.

2.1 General Purpose Graph Visualization Tools

Representation of a large dynamic network in an efficient way is a very challenging problem in scientific visualization domain. A good example is the problem of visualization of the transactions in social networks such as Facebook or Twitter. Although human brain and social networks are quite different in many aspects, such as functional, operational and physical structures, there are some resemblances between them from the network point of view. Both of them are massively parallel and dy-

namic networks. In the human brain, represented by fMRI data, each voxel can be considered as a node whereas, in social networks, each user or page is represented by a node. In human brain functional, structural or anatomical relations can be represented by edges of the network whereas in social networks friendship, following or any interaction between users is represented as an edge in the graph. In both of them, the main difficulty is to visualize thousands of nodes with up to millions of edges.

The tools developed for visualizing the big networks may provide a good starting point for visualization of brain networks. Pajek [11] is one of the good examples of a closed-source stand-alone tool. It was released in 1997 for Microsoft Windows operating systems and still gets updates. Pajek is designed to visualize and analyze large networks, such as; social networks (Facebook, Twitter), citation and co-authorship networks, protein networks etc. The main goals of the tool are decomposition of larger networks into smaller networks to support abstraction, analysis of large networks with a selection of efficient algorithms and providing the user with useful operations to handle the network. Some of these operations are extracting subnetworks, shrinking selected parts of networks, searching for connected components, searching for shortest paths and k-neighbors, computing centralities of nodes and topological properties of networks (degree, closeness, betweenness, hubs and authorities, clustering coefficients, Laplacean centrality).

The main advantage of Pajek is to visualize networks up to 10 billion nodes. In addition, the tool could export the results in several 2D and 3D formats for other special viewers and image editors. Of course, to visualize networks like those, the user should fulfill the hardware requirements (e.g., for a network with around a billion nodes with no edges requires 128 GB or more available RAM).

Although it is one of the best general purpose graph visualization tool available in the literature, Pajek has some flaws when it comes to brain visualization. Firstly, 3D graph drawings of Pajek is not suitable for 3D connectome visualization. As can be seen in Figure 2.1, the nodes are connected with very thick edges. If we try to visualize the brain network with edges and voxels with this size, we most probably see a bunch of intersected pipes. The developers of Pajek solved the hairball problem by

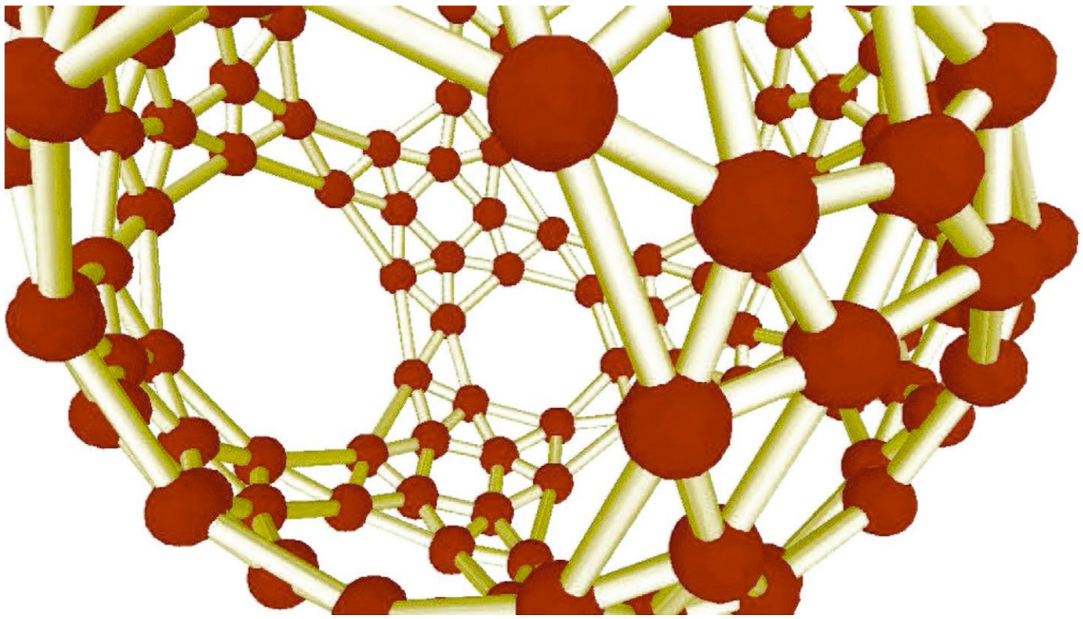


Figure 2.1: 3D graph visualization by Pajek. The layout is obtained by VOS mapping. The image is taken from [11]

employing some layout algorithms such as VOS mapping, drawing in layers, FishEye transformation, Pivot MDS, Fruchterman Reingold optimization and Kamada-Kawai optimization. These algorithms, mainly, place the similar, more connected nodes closer and separate the less connected ones. Therefore, discrimination of different groups becomes easier and edge intersections become less likely. However, replacing the voxels according to their connections, functional regions or any other properties is not possible and allowed in the human connectome. Another problem is that there is no option for loading and visualizing the voxel intensity and edge weight estimations. Instead, the user may employ a limited number of clusters to analyze the graph, which is not useful in observing the intensity change or connectivity alterations in time. Finally, the complex interface and usage steps of the tool. The user should follow a number of operations to get a "meaningful graph", which enables scientists to observe some properties of the graph, such as, hub nodes and regions or denser parts of the graph in terms of connection. Although this gives an enormous graph processing power to the tool, it may not be useful for a neuroscientist.

Another available tool, named JUNG [49], is a Java-based library for modeling, anal-

ysis and visualizing the data. JUNG supports both static and dynamic networks that can evolve by time. Since it is written in Java, JUNG allows users to make use of built-in capabilities of Java Application Programming Interface (API) and other third party Java libraries. One of the main features of the JUNG is the capability of visualizing multi-modal graphs, graphs with parallel edges and hypergraphs. The other important point is having implementations of a number of algorithms from graph theory, exploratory data analysis, social network analysis and machine learning. These algorithms can be used for generating random graphs, clustering, decomposition, optimization, statistical analysis, calculation of network distances and ranking measures (centrality, PageRank, HITS, etc.). Similar to Pajek, JUNG offers some layout algorithms to arrange the position of nodes in the graphs and filtering mechanisms to extract smaller subsets from a larger one.

Since it is written in Java, multiple operating system support is granted. In contrast to Pajek, JUNG is open source and not a stand-alone tool. Therefore, it requires Java API installation before running it. It can be thought as an external Java library. However, to generate the graph, the user should write Java code and that is a difficult task for most of the neuroscientists (and even computer scientists, when the data gets more complex). Moreover, the latest version of JUNG is released in 2010. Since the library and related web page are not updated since then, it lacks developer support and some of the supporting documents are not available anymore.

visANT [32] is another network/pathway analysis and visualization tool. The tool is specifically designed for the integrative visual data-mining of multi-scale Bio-Network/Pathways. The tool runs on the web and handles the social network sized graphs. The tool is capable of network construction with integrated disease and therapy hierarchies, disease-gene and therapy-drug associations and drug-target interactions. Furthermore, visANT uses Network Module Enrichment Analysis (NMEA) to detect phenotypic differences between two networks. In other words, gene expression of a disease can be compared with a healthy one. It can detect some patterns in the larger networks and uses some other graph theory algorithms to process the data. The main difference between visANT and other mentioned tools is weighted network

support. In other words, the tool uses edge thickness, edge color or both to illustrate the weight of edges.

Although visANT seems to be one of the best options for brain visualization, it does not support 3D visualization. It employs layout algorithms to place nodes of the graphs, which are prohibited in connectome visualization. Although it supports many network types, most of its features are very specific to genome analysis, which are useless in other areas.

Unfortunately, most of the available graph visualization tools lack user interaction which is a very crucial task for analyzing the human brain. Additionally, they only display a 2D representation of the graphs. However, the voxels in fMRI data have a specific location in a three-dimensional space and time-varying activity records. A brain network visualization tool should enable the researchers to interactively study and analyze the dynamic changes in the brain network. Thus, although there are some applicable features of social network visualization tools, representing a brain state by a network requires additional capabilities, such as user interaction to facilitate the neuroscientific knowledge in four-dimensional space (x, y, z coordinates and time) and specific graph processors to display a "meaningful graph" and support multi-modal data. In our case, multi-modality refers to different information types can be represented by voxels, such as intensity, node degree, anatomic or functional region.

2.2 Brain Visualization Tools

When a large number of voxels and their interactions are considered, visualization task requires a "meaningful" simplification not only to reduce the computational complexity but also to display the data in a comprehensive way. In recent years, some restricted tools for visualizing the activities in the human brain have been developed. Among them, the most widely used ones are BrainNet Viewer [77] and Connectome Viewer Toolkit [25].

BrainNet Viewer has the ability to show brain network by using nodes and edges in

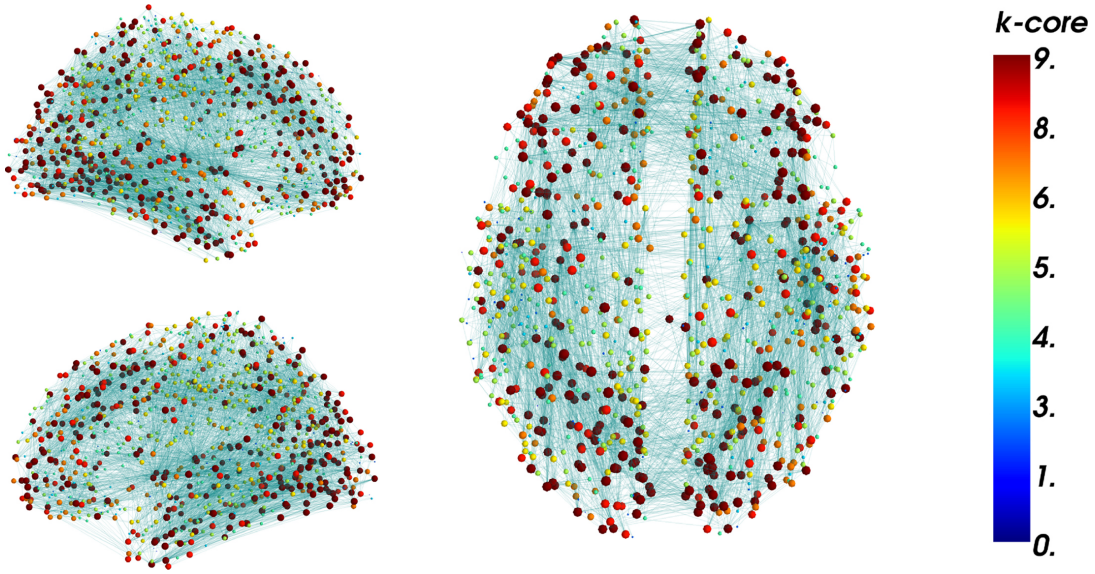


Figure 2.2: A view from Connectome Viewer Toolkit. Size and color of the nodes represent their k-core values computed by NetworkX. The image is taken from [25].

a brain template. The nodes represent the small regions from the brain and the edges are relations between those nodes. The toolbox also displays the region of interests and anatomical regions by mapping the labels to color codes. The user can observe the surface activity by surface mapping view which looks like a heat map on brain template. In addition, this tool enables users to analyze multi-subject experiments on a single screen easily. The user can observe up to eight different brain images at the same time on the screen.

The Connectome Viewer Toolkit, on the other hand, has the capability of displaying more nodes and connections in a brain template than BrainNet Viewer. This tool also views the brain fibers obtained from Diffusion Tensor Imaging (DTI) as well as the tissue group activities. In addition, Connectome Viewer Toolkit combines the power of graph analyzing and visualization libraries such as Mayavi [58], IPython [57] and NetworkX [65]. However, these tools have some limitations and absence of important features, such as, user defined degree and type of simplification which restricts the need and interaction of the researchers.

In Connectome Viewer Toolkit there is no information about the voxel intensities or edge weights between voxels on the graph view. Also, the generated image is not on

voxel level, the nodes in the image represent the brain regions. This approach costs to information loss when the relations between cell groups in the same functional or anatomical region are examined. Connectome Viewer Toolkit provides a static brain grid and colored segments according to the node degree or k-core values (Figure 2.2) in that segment. The tool also enables users to track brain fibers. However, these fibers do not present the weights of the connections between neuronal elements. In addition, requisite input files should be in toolkit specific format which is not commonly used by neither computer scientists nor neuroscientists.

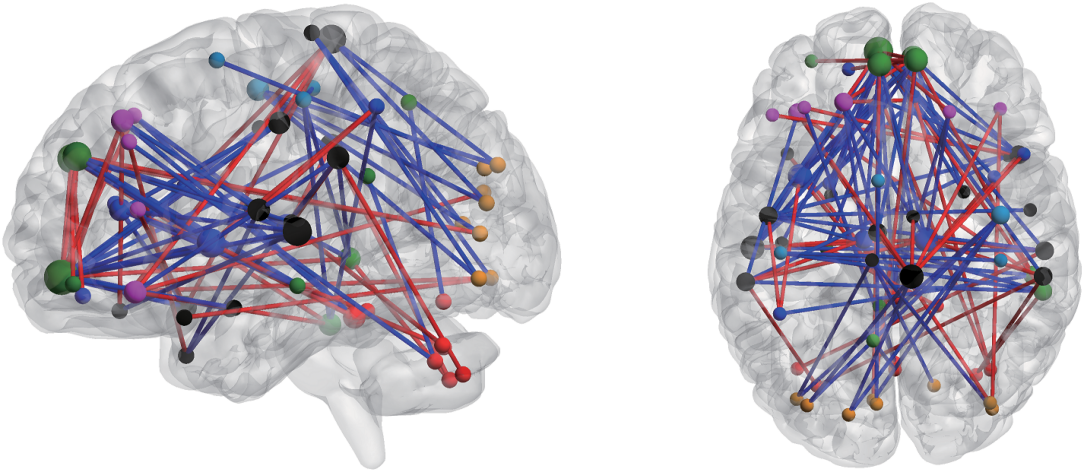


Figure 2.3: A view from BrainNet Viewer where the nodes are colored by their anatomical places. The edge color represent the trend in their connection weight in this particular example. Red lines indicates increased functional connections and blue ones indicates decreased functional connections. The image is taken from [15].

BrainNet Viewer fulfills some of the shortfalls of Connectome Viewer Toolkit. It shows the nodes and edges in a 3D brain template. Similar to Connectome Viewer Toolkit, the nodes are designed to show brain regions or region of interests. Thus, loss of information is still an unsolved problem. The colors of nodes do not represent any specific information such as average intensity value, functional or anatomical region information. The color code of each node is given as an input by the user to represent only one feature of the multi-modal data, as seen in Figure 2.3. In addition, the toolkit does not allow users to simplify the data. Consequently, when the user loads data which contain a high degree of nodes, for example, 20,000 nodes with an

average degree of 40 or 50, displaying the network does not allow for any sensible interpretation. Moreover, one of the main drawbacks of this toolkit is that it is designed and implemented in MATLAB. This results in performance issues when high dimensional matrix operations are considered.

2.3 Nature of fMRI Data

fMRI measurements consist of two types of experiments. Measuring the brain activities during the resting state is one of them and is used to measure the background activities of the brain when the subject is resting in the fMRI machine. On the other hand, in the task-based experiments, the subject is exposed to a stimulus to measure the BOLD response of brain during a cognitive state [48]. In both experiments, the fMRI machine generates sliced shots of the brain in every 2-3 seconds and forms the 3D brain volume for each shot at that time instance t, where $t = 0, \ldots, T$ by registering and stacking the 2D slices.

We represent the intensity of a voxel as $v(t, s_j)$, where s_j is a three-dimensional vector indicating the position of a voxel in the volume in a time instant t. In a typical task-related fMRI experiment, the subjects are exposed to some task specific stimulus in several runs where each run corresponds to a cognitive state. In machine learning terminology the cognitive states correspond to distinct classes as shown in Fig. 2.4. Within large amount of samples across time, only a few of them are assigned class labels (orange vertical lines in Fig. 2.4).

2.3.1 Preprocessing the fMRI Data

When working on fMRI data, it is typically assumed that all of the voxels in a subject's brain volume were acquired simultaneously. In addition, it is assumed that none of the voxel's position has changed during the experiment, in other words, the subject did not move during the measurements. Furthermore, when performing multi-subject analysis, each brain volume is assumed to have the same template so that each subject

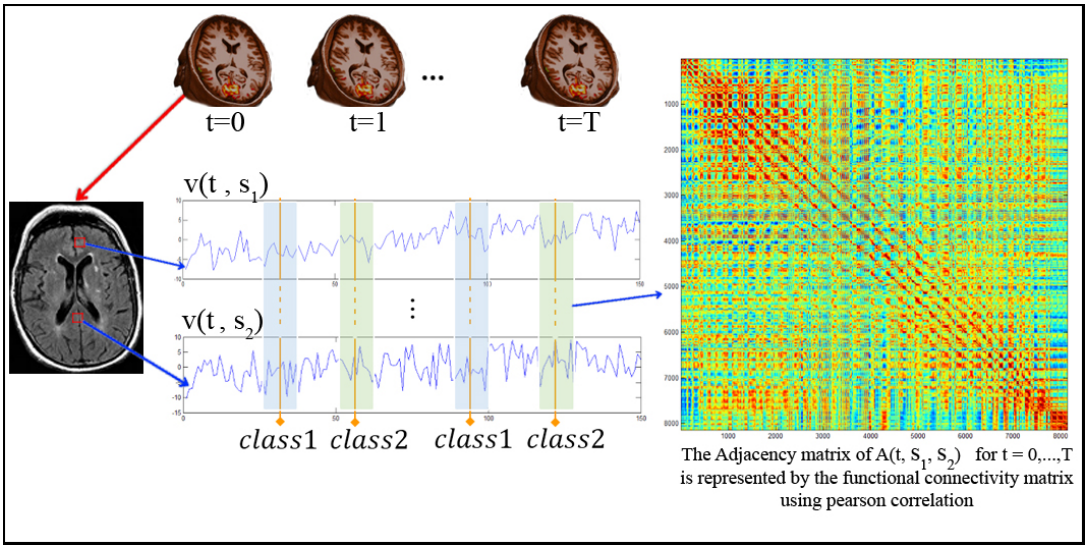


Figure 2.4: The output of a typical fMRI experiment for brain state decoding. 4D fMRI data consist of several volumes across time (left), some of which have assigned to a class label, which corresponds to a specific cognitive stimulus. The vertical orange lines in the time axis (middle) indicate the time instances where the subject is exposed to a new stimulus. For each class, a functional connectivity matrix is formed by considering a suitable time window that encapsulates indicated class labels (right). Colors of functional connectivity matrix show the quantized intensity values, red corresponding to high and blue corresponding to low-intensity values. The image is taken from [46].

has the same number of voxels in their brain volumes and each corresponding voxel between the brain volumes are in the same anatomical locations.

However, in reality, in order to make assumptions valid, some pre-processing on the data should be done. The major operations are slice timing correction, realignment, coregistration of structural and functional images, normalization and smoothing.

During fMRI experiments, the machine takes the whole brain image as sequential 2D slices and obtains all slices to form a complete brain volume which is taken approximately in every 2 or 3 seconds. This results in temporal dislocation between slices. Slice timing correction shifts each voxel's time course to make each slice measured simultaneously.

Another important, pre-processing operation is motion correction. A typical fMRI experiment lasts 20 minutes, and during the experiment even a small movement of the subject results in change of voxel positions. Therefore, when it is not treated carefully, the signal from a specific voxel will be mixed and corrupted with neighboring voxels. In motion correction step, the best possible alignment between the input and reference image is found by rigid body transformation. The aim is to minimize the cost function by tuning the 6 parameters (x, y, z directions and roll, pitch and yaw).

Although they are recorded using the same machines, fMRI provides relatively low spatial resolution compared to MRI, due to the trade-off between time versus spatial resolution, extracting anatomical details becomes harder on fMR images. In order to solve this issue, the raw output from fMRI is mapped onto a high resolution MR image. The process of alignment between these two images is called coregistration. This process can be done by using rigid body or affine tranformation.

As our physical characteristics differ from person to person, our brain shape and size differ, also, a great deal. However, for group analysis it is important to work on a fixed brain structure where each voxel placed in the exact same place across the subjects. In order to do so, regularities shared by a wide range of healthy brains are employed. Each subjects' anatomy is registered to a standardized space defined by a template brain. This process is called normalization and two common templates are used to perform this operation, namely, Talairach and Montreal Neurological Institute (MNI) templates. In this thesis, we have mainly focused on the latter one to place anatomical mappings on a standard template.

In the final stage of pre-processing, a spatial smoothing should be applied to fMRI data prior to further analysis. Smoothing is done via convolving the fMRI images with a Gaussian kernel. This process improves inter-subject registration and decreases the effects of normalization.

More details about functional magnetic resonance imaging and processing can be found in Lindquist's work [39].

2.3.2 Brain Connectivity

The human brain is inherently a very large scale dynamical network that we can call it as the universe in our heads. However, human brain differs from the universe by a very crucial structure: Unlike the galaxies, neurons are physically connected to each other and they are in communication to enable different sensorimotor and cognitive tasks to be performed [31]. Recent studies in neuroscience and cognitive science show that the connectivities among the voxels or anatomic regions carry important information about the underlying cognitive process more than the voxel-wise or regional activities in the human brain [61, 56, 60, 19, 7, 20, 50]. These connections can be categorized into three types, namely, anatomic, functional and effective connectivity.

Anatomic connectivity, or sometimes called structural connectivity, refers to a network of physical (synaptic) connections between sets of neurons. Functional connectivity is based on the experimental evidence that neurons with similar activity contributes to the same cognitive task. It is expected that voxels with similar time series form the underlying task. Functional connectivity is measured by statistical correlations. In other words, functional connectivity captures deviations from statistical independence between distributed and often spatially distant neuron groups. Effective connectivity can be thought as the intersection of anatomic and functional connectivity. The effective connectivity describes the causal effect of one neuron or node in the connectome to another [23].

Both functional and effective connectivity estimates the functional similarity of nodes or regions by using different metrics. In order to measure the similarity between two neuronal elements, their intensity fluctuations across the time are compared. Therefore, whichever metric is used, the input data is always the same, which is voxel time series. Although there are many statistical methods proposed and used in the literature [22, 61, 7, 56, 19], we have only used following two methods to measure the connectivity among the voxels in our proposed tool, CEREBRA.

The first one is the zero-lag Pearson correlation, which constructs the functional connectivity between the voxels. The Pearson Correlation between two voxel pairs i and

j is defined as follows:

$$\rho(i,\, j) = \frac{cov(v(t,\, s_i),\, v(t,\, s_j))}{\sigma_{v(t,\, s_i)} \times \sigma_{v(t,\, s_j)}}, \tag{2.1}$$

where $cov(v(t,\, s_i),\, v(t,\, s_j))$ is the covariance between the voxel intensity records of the voxels i and j in time ($v(t,\, s_i)$ and $v(t,\, s_j)$). The $\sigma_{v(t,\, s_i)}$ and $\sigma_{v(t,\, s_j)}$ are the standard deviations of the intensity records for the voxels i and j respectively. The functional connectivity matrix with the entries of $\rho(i,\, j)$ is assumed to be the network adjacency matrix, $A(t,\, s_i,\, s_j)$, for each time interval starting at the time instance t.

The other method used in this thesis is Local Mesh Model with Temporal Measurements (LMM-TM) [51], which calculates the effective connectivity between neuronal pairs. The proposed method models the relationship among voxels residing in a spatial neighborhood using the time series of intensity records, BOLD responses, measured during a cognitive process. In order to estimate the relationship, the algorithm forms a local mesh around each voxel, called seed voxel. Then, a mesh is constructed over a neighbor system defined for the seed voxel where the mesh arc (edge) weights are estimated using a linear model of measurements of BOLD response recorded during the presentation of a stimulus. In each local mesh, the intensity of the seed voxel is represented by a weighted linear combination of the nearest adjacency voxels' intensity records. This method is based on Bazargani et al.'s work [12], which states that voxels belonging to a homogeneous region of interests have the same Hemodynamic Response Function (HRF) shape with varying amplitude. Onal et al.'s work, in which they define a formal and detailed representation of a voxel intensity by its neighbors, could be simply defined as follows:

$$\bar{r}(t, s_j) = \sum_{s_b \in \eta_p} a_{t,s_j,s_b} \times \bar{r}(t, s_b) + \bar{\epsilon}_{t,j}, \tag{2.2}$$

where, $\bar{r}(t, s_j)$ is the BOLD response of the voxel placed at $s_j = (x_j, y_j, z_j)$ for the sample taken at t and η_p is the p nearest neighbor list of that voxel. The relation between the voxel pairs are indicated by a_{t,s_j,s_b} in the equation. The error in this

estimation is represented by $\bar{\epsilon}_{t,j}$ and defined as,

$$\bar{\epsilon}_{t,j} = (\epsilon_{1,j}, \epsilon_{2,j}, ..., \epsilon_{T,j})^{\top}. \tag{2.3}$$

In the Eq. 2.3, the error is measured for each time instant $t = 1, 2, ..., T$. The adjacency matrix $A(t,\ s_i,\ s_j)$, is filled by the estimated values of a_{t,s_j,s_b} for each seed voxel and for each time instance t.

2.3.3 fMRI Network Representation

The most fundamental contribution of this study is to represent the human brain as a three-dimensional time varying dynamic network, extracted from the fMRI data. Therefore, the information embedded in the fMRI data should be represented in terms of a dynamic network as;

$$D(t) = \{v(t,\ s_j),\ A(t,\ s_j,\ s_k) :\ j,\ k = 1, \ldots, N\}, \tag{2.4}$$

for each discrete time instant t where $t = 1,\ \ldots,\ T$. Each node corresponds to the voxel at location $s_j = (x_j,\ y_j,\ z_j)$ at a time instant t with intensity value, $v(t,\ s_j)$, as mentioned before. The entries of the network adjacency matrix $A(t,\ s_j,\ s_k)$, represent the directed edge weights from voxel j to voxel k at a time instance t. They are estimated by using spatial or functional neighboring methods described above. Fig. 2.4 shows an example of the graph adjacency matrix $A(t,\ s_j,\ s_k)$ extracted by using functional connectivity matrix obtained by computing zero-lag Pearson correlation.

Please note that, a user can employ any edge-weight estimation algorithm as long as it generates the network data structure $D(t)$, for $t = 1,\ \ldots,\ T$.

2.4 Connectivity Reduction Methods

A typical fMRI experiment brings $100{,}000$ to $200{,}000$ voxels to be displayed [40]. Therefore, a simple Pearson Correlation calculation brings 10^6 edges to display, if each voxel is limited to have ten outgoing connections. Displaying this number of edges causes the hairball problem. In order to avoid this problem, basic thresholding and edge reduction algorithms can be applied to the human connectome. In this section, two of these methods are explained in separate sections.

2.4.1 Basic Thresholding Methods on Edge Weights

As mentioned above the initial graph may have 10^6 or more edges on the display. However, only the small percentage of them contains crucial information, related to the underlying cognitive process. Therefore, only a specific group of connections should be monitored to observe the patterns of connectivity. The pruning operation can be done by basic thresholding methods by assigning upper and lower weight value bounds. They are simple, easy to implement and gives good results on many application areas, such as observing the edges between highly correlated voxels when using Pearson correlation or observing the edges between voxels that have the similar HRF functions in the LMM-TM method. We have embedded three different thresholding methods into CEREBRA which are explained below.

- **High-pass Thresholding:** A high-pass threshold passes edges with higher weights than a certain cutoff weight and attenuates the signals with lower weight values than the cutoff value. Thus, this thresholding attenuates the connections between less correlated voxels and leaves the strongest connections on the graph.
- **Low-pass Thresholding:** A low-pass threshold passes edges with a lower weight value than a certain cutoff weight value and attenuates the edges with higher weight value than the cutoff value. When it is used on the human connectome, it eliminates the edges with mid and high weight values as leaves the

weakest connections.

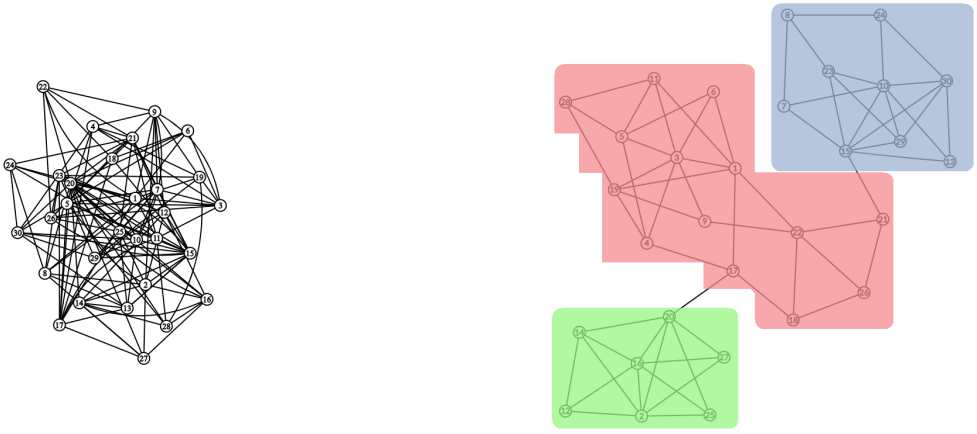
- **Band-stop Thresholding:** A band-stop threshold blocks the weight values within a certain range and passes the edges with weight values above or below that range. In other words, in a typical usage, it eliminates the mid weights while allowing the high and low ones. When applied to the brain graph, it leaves the strongest and weakest connections, in terms of weights, while attenuating the connections with mid valued weights. Therefore, this allows users to observe the edges between most negative and positive correlated voxels while hiding edges between uncorrelated or less correlated voxels.

2.4.2 Local Graph Sparsifier

This method is suggested by Satuluri et al. [63] for undirected graphs. The main purpose of this work is to reduce the number of edges while preserving the underlying structure of the graph. By preserving the underlying structure of the graph, we mean preserving the overall shape and structure of the connections in the graph. In addition, preserving the structure of the graph while reducing the number of edges will enable scientists to run graph segmentation algorithms (classifiers, clustering algorithms) faster without any sacrifices on accuracy.

In the proposed work, their attempt is to reduce the graph size in order to scale up community discovery/graph clustering algorithms. The key point is, this reduction algorithm only filters out the edges in the graph while preserving all the nodes. This way, the groups formed by edges in the graph is retained and may even be enhanced in visual aspect as observing the groups will become easier when the number of edges is reduced. An example is visualized in Figure 2.5a, where the original graph has the hairball problem on a small scale. The graph has 30 nodes and 128 edges belonging to three different node clusters. The sparsified graph, shown in Figure 2.5b, has only 64 edges and the hub nodes of the graph are much clearer in this new graph. Please note that, in Figure 2.5b, the nodes are re-aligned according to their connections.

The main difference between this algorithm and the basic thresholding defined in



(a) Original Graph

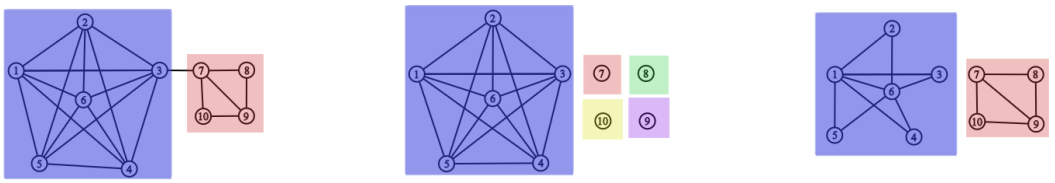
(b) Sparsified Graph

Figure 2.5: An artificial demonstration of the local sparsification method with 30 nodes and 128 edges. The sparsified graph only has 64 edges.

the Section 2.4.1, is that this algorithm assigns a local threshold value for each node whereas the basic thresholding applies the same global threshold value for every node. Setting a global threshold value is problematic since each cluster has different densities (number of within connections) and this type of threshold may leave more edges in the denser clusters as disconnecting the sparser ones. The difference between local and global thresholding methods is illustrated in Figure 2.6. Figure 2.6a shows the original graph and Figure 2.6b and Figure 2.6c are the sparsified graphs. The graph in Figure 2.6b is a likely output of applying a global thresholding method, where the inner connections in the small cluster are suppressed. On the other hand, applying a local threshold value, as in Figure 2.6c, ensures that the inner connections in clusters are preserved.

As can be seen from the figures mentioned above, the algorithm mainly suppresses the inter-cluster edges, edges between clusters. Meanwhile, it holds the intra-cluster edges, edges within a cluster. In this way, we can be sure that the cluster structure of the original graph is retained. The proposed algorithm looks for the similarity between the nodes to decide whether the edge between those nodes should be suppressed or not. An edge between the nodes i and j is likely to lie within a cluster if the nodes i and j have similar adjacency lists (common neighbors). As a contrary, the

edge between the nodes i and j is likely to be an inter-cluster edge if the adjacency lists of the nodes i and j have less common neighbors in their adjacency lists.



(a) Original Graph (b) Global Thresholding (c) Local Thresholding

Figure 2.6: Comparison between global thresholding and local sparsification. Since global thresholding aims only to remove a certain number of edges from the graph, it may result in removing all edges from the same area which changes the structure of the original graph. On the other hand, local thresholding ensures that each node will have at least one connection. Consequently, local thresholding method preserves the original structure which has two clusters whereas the global thresholding method increases the number of clusters to five by destroying the small cluster in the original graph.

The main assumption of Satuluri et al. is that the more common neighbors have the nodes i and j, the more likely they lie in the same cluster. Therefore, they have used the Jaccard measure to quantify the overlap between adjacency lists. Let $Adj(i)$ be the adjacency list of the node i and $Adj(j)$ be the adjacency list of the node j. Then, the similarity between the adjacency lists of these nodes, and implicitly between nodes themselves, is defined as follows:

$$Sim(i,\, j) = \frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|}. \tag{2.5}$$

Using the equation above, the algorithm ranks each edge according to their similarity for each node i. Then, the algorithm chooses the top $f(d_i) = d_i^e$ edges incident to node i with degree d_i where e is the global sparsification ratio that helps control the overall sparsification ratio of the result. The higher values of the e result in a denser graph and lower values of the e result in a sparser graph. The value of e should be in between $[0,\, 1]$ interval. Sorting and thresholding the edges of each node

separately allows the sparsification to adapt to the densities in that specific part of the graph. Additionally, this procedure ensures that the algorithm picks at least one edge incident to each node.

Although the Jaccard similarity metric performs well on undirected and unweighted graphs (e.g. Wikipedia, Human-PPI, Flicker) as Satuluri et al. show in their work, in brain networks, mixing the direction of the edges under the same metric spoils the consistency of the edges in a specific direction, resulting in the removal of some crucial edges in both direction. On the other hand, the connections between voxels have both direction and weights to be considered while applying the sparsification process.

2.5 Chapter Summary

In this chapter, popular general purpose and brain specific visualization tools are discussed. Then, a brief information about fMRI data and information about how do we map fMRI data into a graph structure are given. This is followed by preprocessing steps of the fMRI data to give more insights about the construction of the input data. The chapter continues with explaining how the connections between voxel pairs are established and which methods are used in this thesis. As the final point, we have discussed the methods to reduce the number of edges in a dense graph. The key points of this chapter are:

- Human connectome can be represented as a 3D dynamic graph structure. However, it needs a special treatment when it comes to visualization of this graph. Although the connectome itself is not a big structure when compared with social networks or any other web data, general purpose graph visualization tools are not sufficient to a detailed analysis of it.
- Currently available brain visualization tools cause information loss by not displaying all the voxels and reflecting multi-modality of the data. Also, they either need program specific input files or have performance issues.

- The fMRI data should undergo some pre-processing operations before the input data takes its final form. These operations correct the defects occurred over the course of the experiment to a certain extent.
- The fMRI data can be represented as graph structure in which the actual data comes from the machine considered as nodes, voxels, of the graph. Edges could be estimated by an independent algorithm such as Pearson correlation or Local Mesh Model.
- When the number of edges is too high, visualizing such a graph creates hairball problem. In order to avoid this problem, the graph should be sparsified, in terms of edges.

CHAPTER 3

INPUT FILE STRUCTURE AND PRE-PROCESSING OF FUNCTIONAL MAGNETIC RESONANCE IMAGING DATA

Recall that functional magnetic resonance imaging (fMRI) data consists of hundreds of thousands of voxels, each of which has a time series of length T, which is the duration of the cognitive experiment. Furthermore, the voxels are massively interconnected. Therefore, the input file structure of such a data requires a crucial design to improve the efficiency of the processes of data.

This chapter provides information about input, the file structure of fMRI data and reading process. The chapter begins with explaining the supported input file formats and how the data is contained in the files. In this section, each part of the input files is explained and detailed. Then, the chapter continues with data storage in CEREBRA. In other words, defining the class structure that holds the data. We conclude this chapter by explaining the pre-processing steps applied on input data processed at the beginning of CEREBRA routines.

3.1 Input File Types of CEREBRA

The developed tool supports two commonly used file types in this research area. One of them is .mat file type, which is a typical output of any MATLAB code. Many pre-processing operations are done using MATLAB, therefore, reading input from a .mat file will definitely ease scientists' work as they can visualize their data without any need to convert it into another file type and, most of the times, format. The other

supported file type is the plain text file format, which is one of the simplest formats. Initially, this feature is added for test purposes and then left as an additional feature to encourage researchers who want to visualize their data without any knowledge of coding, file types, and structures. This section gives the details of these two file formats, how we read them and how they are structured.

3.1.1 Mat File

As mentioned before, CEREBRA has been implemented using C++ as the main programming language. In order to read and process .mat files in C++, we have employed MATLAB MAT File I/O Library (Matio) [3]. Matio is an open-source C library for reading and writing binary MATLAB MAT files. With the help of this library, we do not have to access and use MATLAB's shared libraries. Therefore, it annihilates the MATLAB dependency of the tool.

There are many functions available in the library to read and write specific variable types. We have used double type operations since they are most common and can handle some of the other types (e.g. int and float). In order to read a .mat file, we need to parse it correctly before reading the values in the files. For this purpose, we have used *mat_t* and *matvar_t* classes defined in Matio. The first class contains information about a MATLAB .mat file. We use this definition in order to read specific information about the data such as variable names, rank, size etc. When reading operation returns successfully, we can read the specified field of the loaded .mat file into the *matvar_t* variable. In our case, the information contained in the *matvar_t* is generally a 2D matrix with double typed variables. We can fetch the dimensions of that matrix using *dims* field of the *matvar_t* class and read the actual information (position, intensity, neighbors etc.) using *data* field by iterating it according to dimension sizes.

In .mat format, voxel positions, voxel intensities, edge lists, edge weights and MNI coordinates transformation matrix should be bundled in a single .mat file. Therefore, each of those parameters is a variable in that file. The label information for label view, however, should be given in a secondary .mat file. Also, in order to enable users to

load several label information at once, each label information should be a variable in that file.

3.1.2 Plain Text File

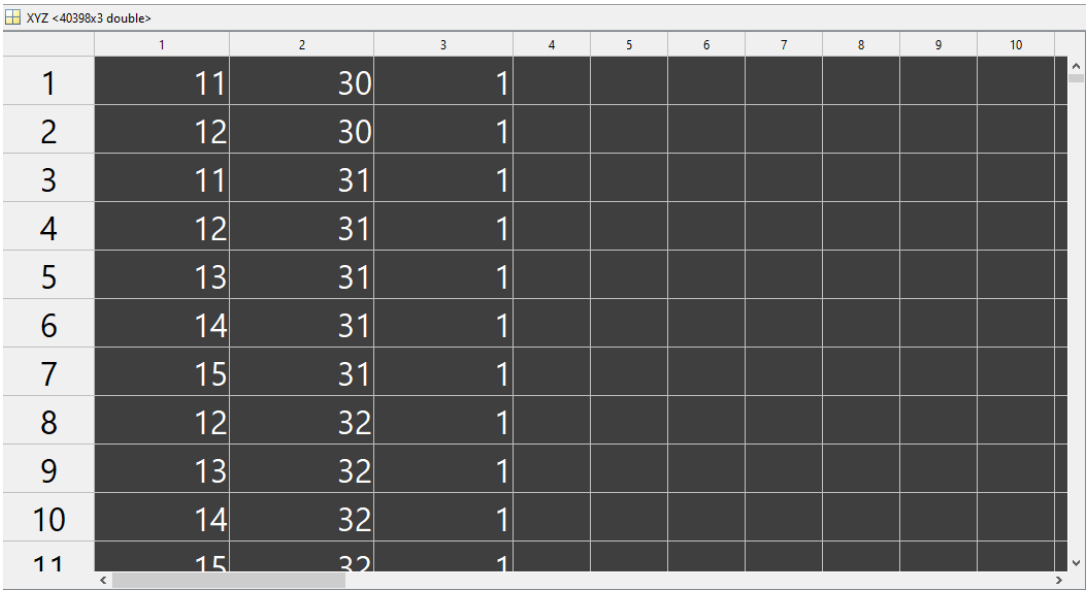
The tool requires a different structure for the plain text files. Each parameter should be contained in a separate .txt file under a shared folder. For instance, voxel positions are stored in "xyz.txt", voxel intensities are stored in "intensities.txt", edge lists are stored in "edges.txt" and their weights are stored in "weights.txt" which are placed in a folder called "ExampleData". **Please note that, the current version of CEREBRA does not support label information in plain text format**. Therefore, even if the user loaded the data in plain text format, the label information should be provided in .mat format.

3.2 Input Structure

In order to display a brain network properly by CEREBRA, some information, such as voxel positions, intensity records, edge lists (neighborhood information), edge weights, MNI coordinates transformation matrix and label information should be provided to the system. Most of this information could be provided in either MATLAB file format or plain text file format. In this section, we will describe how this information should be provided in both file types.

3.2.1 Voxel Positions

Voxel positions should be given in 3D Cartesian space. Thus, each voxel's position is given in x, y, and z coordinates. For this purpose, voxel position information should be in $N \times 3$ matrix format where each row corresponds to a voxel's position and N is the number of voxels.



	1	2	3	4	5	6	7	8	9	10
1	11	30	1							
2	12	30	1							
3	11	31	1							
4	12	31	1							
5	13	31	1							
6	14	31	1							
7	15	31	1							
8	12	32	1							
9	13	32	1							
10	14	32	1							
11	15	32	1							

Figure 3.1: An example of a voxel position variable in MATLAB. This example encapsulates $40{,}398$ voxels. Although this example sorted according to their z, y and x positions, there is no need to sort voxels in any order.

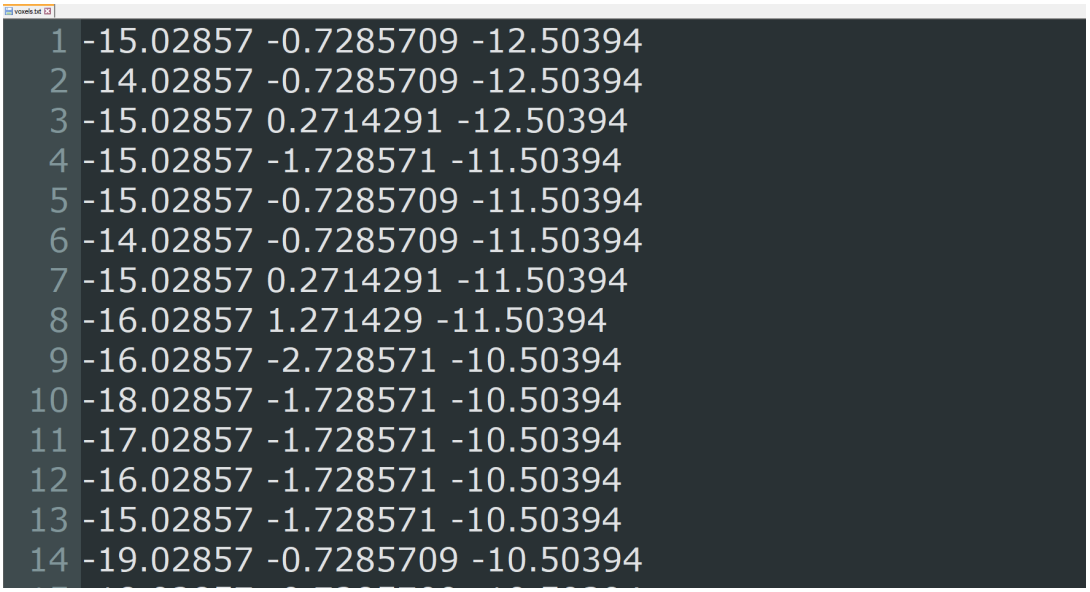


Figure 3.2: An example of a voxel position plain text file. As opposed to the MATLAB version of it, there is no dimension indicator on this format. Each variable should be separated by a single space and each line corresponds to one voxel position.

For the MATLAB file format, this information is given as a variable in the input file. The variable is basically a matrix that fits the structure defined above. The tool does not require any specific name on this variable in this format. An example voxel position variable is shown in Figure 3.1.

For the plain text file format, this information should be placed in a single .txt file under the input folder. There should be N lines in the text for N voxels and in each line there should be three values, separated by a space, reflecting the x, y, and z coordinates respectively. The tool does not require any specific file name as in the .mat format. An example voxel position file can be seen in Figure 3.2.

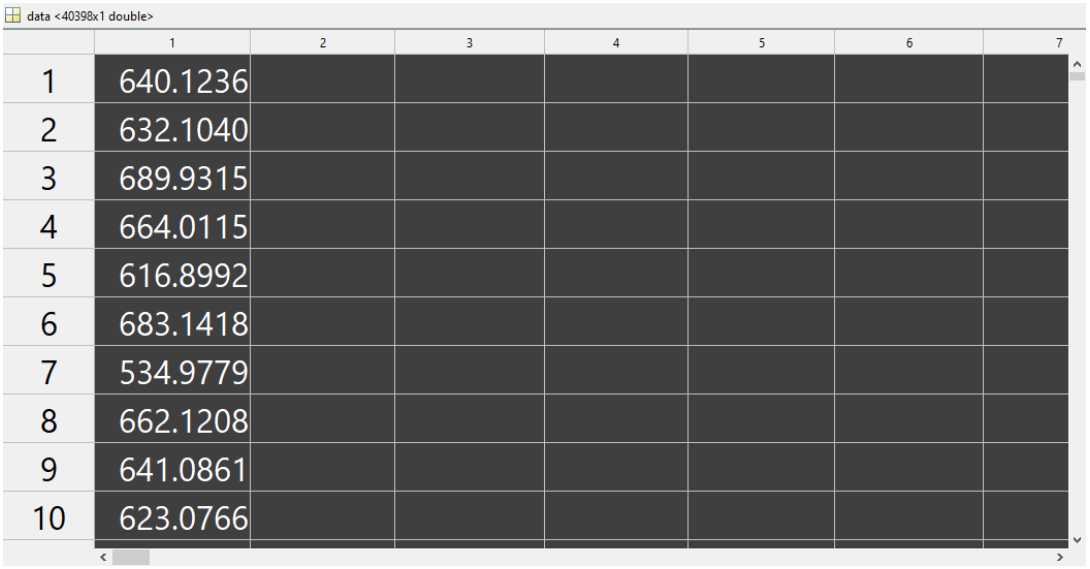
Please, note that the line number of a voxel's position information in both types is related with its *i.d.*. In other words, the first voxel is indexed as 0, the second one is indexed as 1 and so on. This identification helps us to match position information with other ones.

3.2.2 Voxel Intensities

During a cognitive experiment, the fMRI machine takes sliced shots of the brain in every 2 to 3 second and forms the 3D brain volume for each shot at a time instance t, where $t = 0, \ldots, T$. In each time instance t, the machine measures the BOLD response of the brain that we call the intensity record. Intensity records are formed as an $N \times T$ matrix where N is the number of voxels and T is the number of time instances. If T is greater than one, CEREBRA animates voxel colors to simulate the whole experiment.

A row of this matrix indicates the intensity of the voxel positioned at the same row in voxel position variable/file. Therefore, there should be a one-to-one match between this and voxel position variable/file.

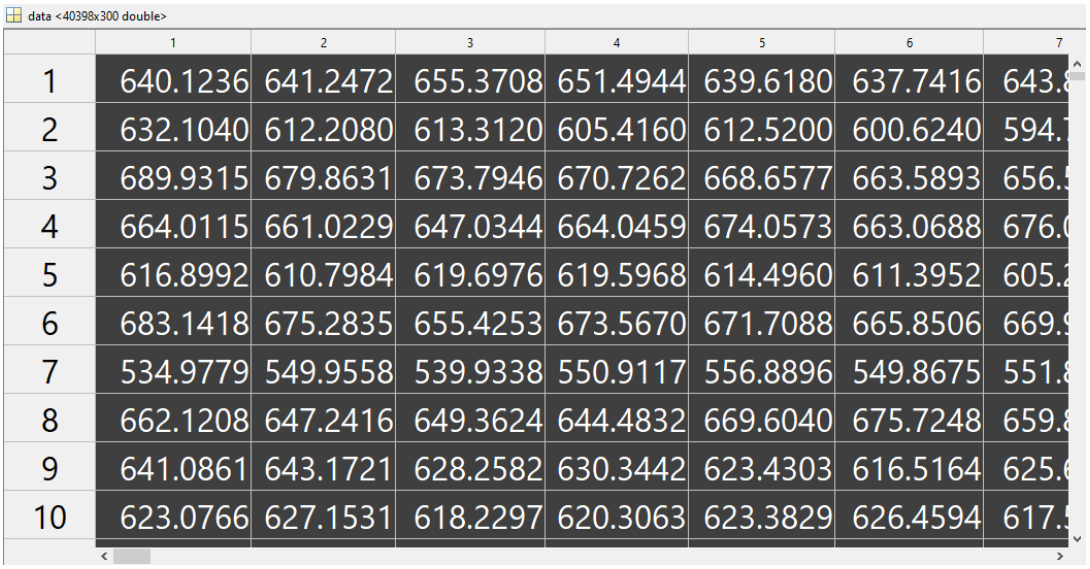
For the MATLAB file format, this variable is a matrix that meets the structure defined above. There is no need to put a specific name to that variable. Figure 3.3 illustrates the variable in both static and time-series perspectives.



data <40398x1 double>

	1	2	3	4	5	6	7
1	640.1236						
2	632.1040						
3	689.9315						
4	664.0115						
5	616.8992						
6	683.1418						
7	534.9779						
8	662.1208						
9	641.0861						
10	623.0766						

(a)



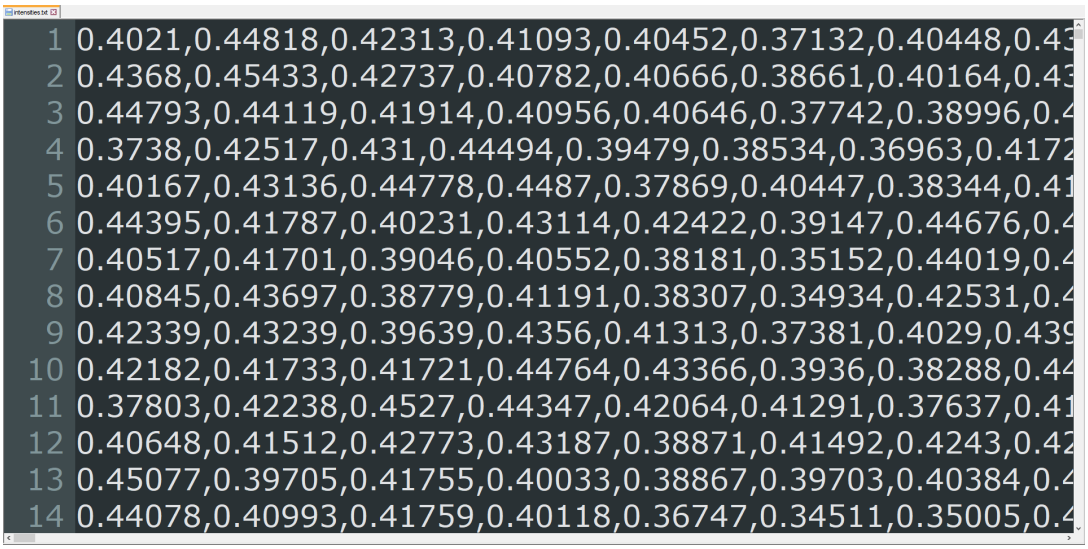
data <40398x300 double>

	1	2	3	4	5	6	7
1	640.1236	641.2472	655.3708	651.4944	639.6180	637.7416	643.8
2	632.1040	612.2080	613.3120	605.4160	612.5200	600.6240	594.7
3	689.9315	679.8631	673.7946	670.7262	668.6577	663.5893	656.5
4	664.0115	661.0229	647.0344	664.0459	674.0573	663.0688	676.0
5	616.8992	610.7984	619.6976	619.5968	614.4960	611.3952	605.2
6	683.1418	675.2835	655.4253	673.5670	671.7088	665.8506	669.9
7	534.9779	549.9558	539.9338	550.9117	556.8896	549.8675	551.8
8	662.1208	647.2416	649.3624	644.4832	669.6040	675.7248	659.8
9	641.0861	643.1721	628.2582	630.3442	623.4303	616.5164	625.6
10	623.0766	627.1531	618.2297	620.3063	623.3829	626.4594	617.5

(b)

Figure 3.3: Two possible forms of voxel intensity variable in MATLAB file format. Figure 3.3a illustrates a case, where there is no time information embedded in the data. In this case, each voxel has a static color on display according to their intensity values. Figure 3.3b is the time series form, prepared for the same data. Each line belongs to one voxel and the elapsed time between each cell is assumed to be two seconds for all experiments. This is because fMRI machines shoot images between 2 or 3 seconds in general.

(a)



(b)

Figure 3.4: Two possible forms of voxel intensity files in plain text format. As in the MAT-File version, each line consists of one value as the corresponding voxel's intensity value in static case (Figure 3.4a). Figure 3.4b is the time series form in plain text format. Each line belongs to one voxel and each time instance is separated by a comma.

In the plain text file format, this information should be stored in a single plain text file under the input folder. There should be N lines each of which is dedicated to a voxel, as in the voxel position text, and in each line, there should be T values separated by commas. The examples for static and time-series versions of this file are given in Figure 3.4.

3.2.3 Voxel Relations (Edge Existence Information)

As mentioned in the Background chapter, recent studies in neuroscience and cognitive science show that the connectivities among the neuronal units carry important information about the underlying cognitive process more than the voxel-wise or regional activities in the human brain. CEREBRA is capable of displaying and animating the connection information to enable scientists to study on these connections and their change during a cognitive experiment.

Please note that, the provided connection information is considered to be directed by CEREBRA. Thus, when the voxel j is in the neighborhood list of voxel i, that means there is an edge goes from voxel i to j. However, this does not imply any information about the existence of a connection from voxel j to i.

In MATLAB file format, this information is embedded into an $N \times 1$ cell-array variable, where N is the number of voxels. Each cell contains the neighbor list of a voxel which is in $M_i \times 1$ array format and where M is the number of neighbors of voxel i, positioned at the $(i-1)^{th}$ line in voxel position variable. The subtraction is done because in MATLAB and plain text files indexing starts with 1 whereas in C++ indexing starts with 0. Therefore, similar to the voxel intensity records, there should be a one-to-one match between this and voxel position variable. An example variable content is given in Figure 3.5.

In plain text format, this information is contained in a single .txt file under the input folder. There should be N lines, where N is the number of voxels, and each line should contain the i.d.'s of that voxel's neighbors separated by spaces as shown in Figure 3.6. As in the figure, each voxel may have a different number of neighbors.

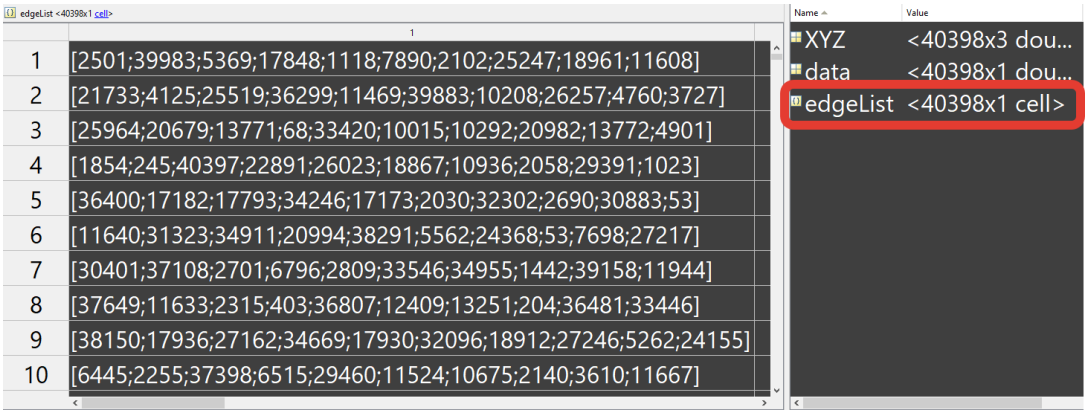


Figure 3.5: An example edge list variable in MATLAB file format. The type of this variable should be a cell list (marked with a red box). Each line corresponds to a voxel's neighbor list. The numbers in the list indicate their line numbers on position variable.

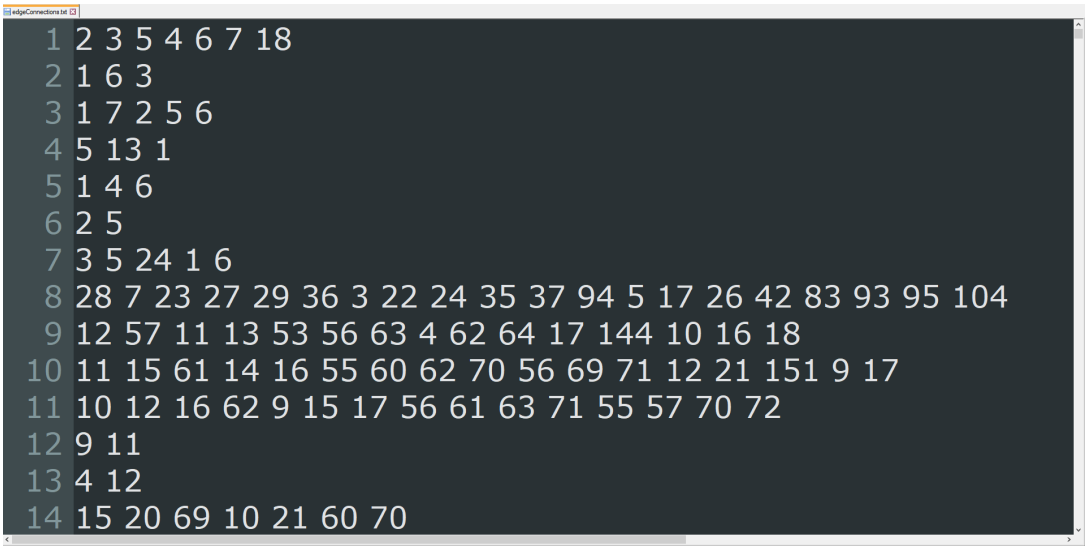
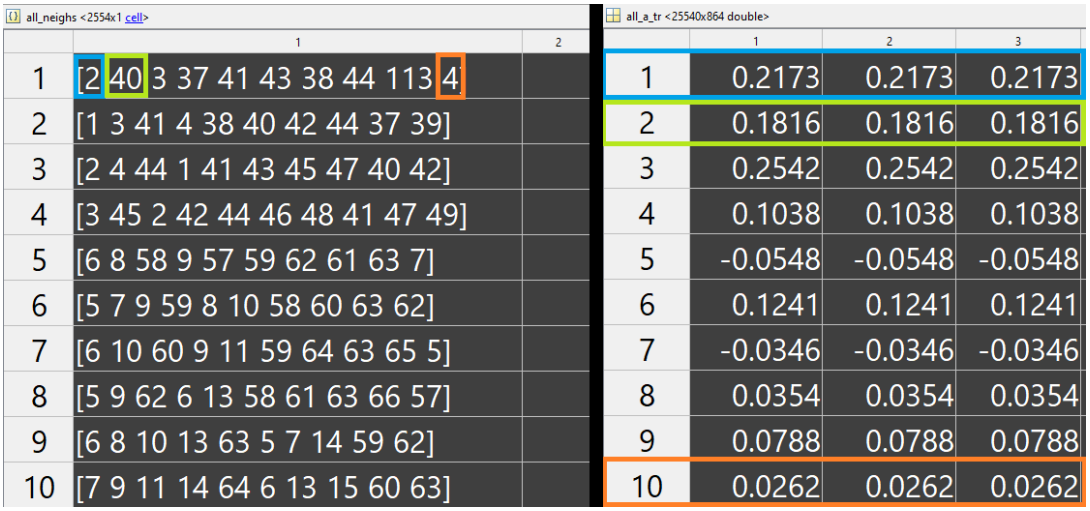


Figure 3.6: An example edge list file in plain text format. Each line belongs to the voxel that is placed at the same line in position file and each number, separated by a space, indicates the voxel that is placed at that line in the position file.



all_neighs <2554x1 cell>

	1	2
1	[2 40 3 37 41 43 38 44 113 4]	
2	[1 3 41 4 38 40 42 44 37 39]	
3	[2 4 44 1 41 43 45 47 40 42]	
4	[3 45 2 42 44 46 48 41 47 49]	
5	[6 8 58 9 57 59 62 61 63 7]	
6	[5 7 9 59 8 10 58 60 63 62]	
7	[6 10 60 9 11 59 64 63 65 5]	
8	[5 9 62 6 13 58 61 63 66 57]	
9	[6 8 10 13 63 5 7 14 59 62]	
10	[7 9 11 14 64 6 13 15 60 63]	

all_a_tr <25540x864 double>

	1	2	3
1	0.2173	0.2173	0.2173
2	0.1816	0.1816	0.1816
3	0.2542	0.2542	0.2542
4	0.1038	0.1038	0.1038
5	-0.0548	-0.0548	-0.0548
6	0.1241	0.1241	0.1241
7	-0.0346	-0.0346	-0.0346
8	0.0354	0.0354	0.0354
9	0.0788	0.0788	0.0788
10	0.0262	0.0262	0.0262

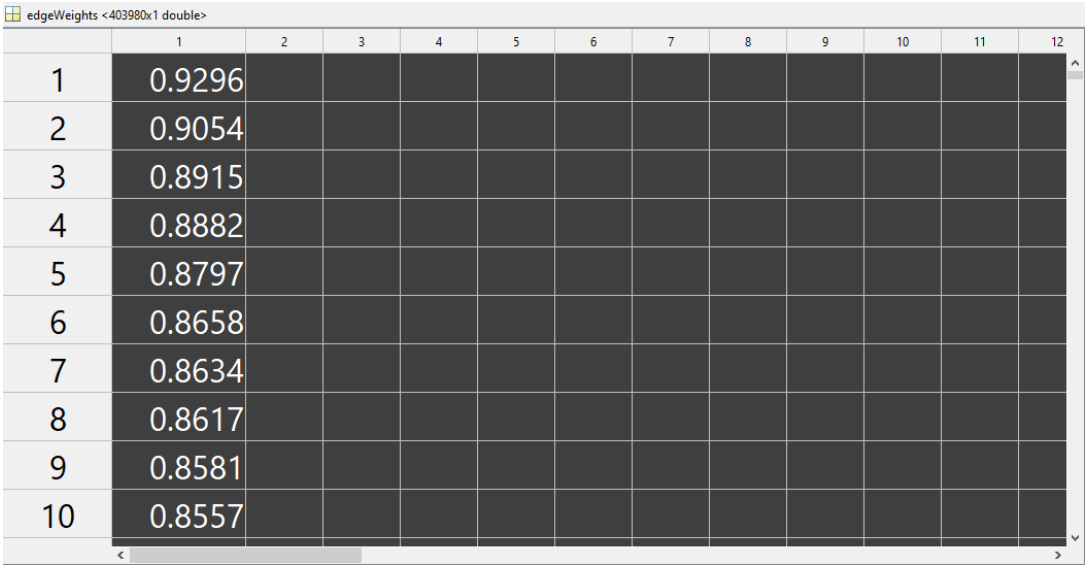
Figure 3.7: The matching between edge existence information and edge weight information. The left side of the figure reflects the neighbor information in which each line contains the neighbors of a voxel in a cell. The right side of the figure is the corresponding edge weight information, where each line carries information about one edge pair. The matching between those two is illustrated by colored boxes.

3.2.4 Edge Weights

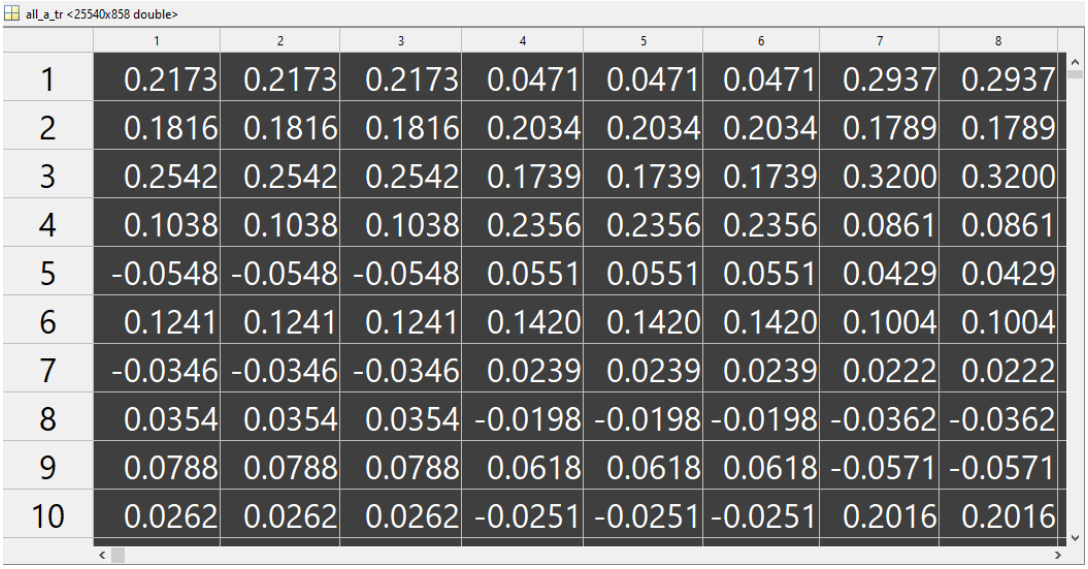
The arc weights between the voxels form a dynamic network, which is subject to change at each time instant as the voxel intensity values change. For this reason, CEREBRA displays and animates the edge weights by coloring the edges according to an assigned color to each volume interval. Edge weights should be in a $K \times T$ matrix format where T is the number of time instances and K is the number of all edges in the graph, which can be calculated as $\Sigma_{i=1}^{N} M_i$. N is the number of voxels and M_i is the number of neighbors of the voxel i.

Each row of the edge weight matrix corresponds to a specific pair's weight value. The first row indicates the weight of the first voxel's first neighbor, the second row indicates the weight of the first voxel's second neighbor and so on. An example matching is shown in Figure 3.7.

In the MATLAB file format, edge weights are given as a $K \times T$ matrix variable, as mentioned above. When T is greater than one, CEREBRA animates the edge colors



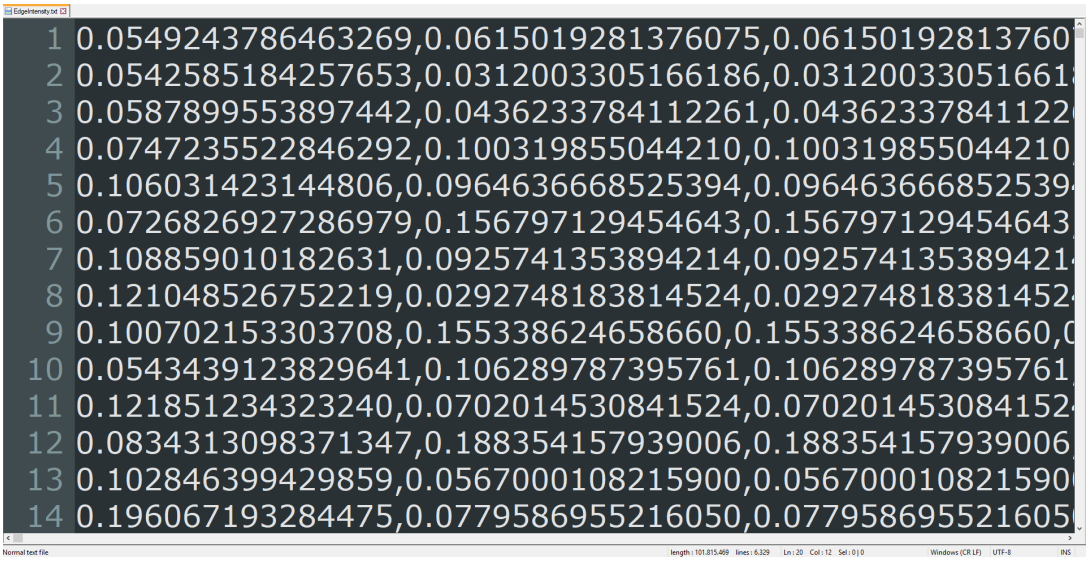
(a)



(b)

Figure 3.8: Two possible forms of edge weight variable in .mat format. In Figure 3.8a each line has one value as the corresponding pair's weight value. Figure 3.8b is the time series form of this variable. Each line belongs to one pair.

(a)



(b)

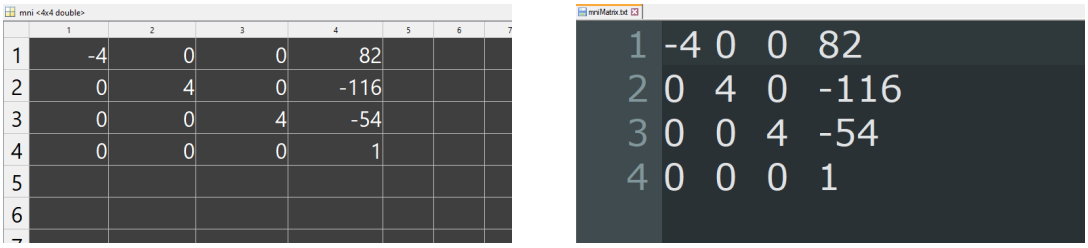
Figure 3.9: Two possible plain text versions of the edge weight information. Figure 3.9a illustrates a static version in which each line holds one value for a specific pair. In Figure 3.9b, each line still belongs to one pair and this pair's weight value changes in time as the values in the same row fluctuating. Each value should be separated by a comma.

according to the values in rows. An example of this variable is shown in Figure 3.8. In plain text file format, this information is given in a .txt file under the input folder, where each line corresponds to one row of the matrix. The entries in the rows should be separated by commas. Content of an example edge weight .txt file is shown in Figure 3.9. Please note that, it is not possible to find which line indicates which pair's weight value just by looking this information. This information should definitely be used with pair information. In addition, as with the other information types, there is no need to put a specific name to this information in both file format types.

3.2.5 Montreal Neurological Institute (MNI) Transformation Matrix

MNI space is used to normalize the wide range of brains which changes, in terms of shape, with respect to each subject from their personal space to a standard one. Although CEREBRA could not compute the necessary transformation matrix, it allows users to provide transformation matrix and switch between personal space and MNI space.

MNI transformation matrix is a 4×4 matrix and is given as a variable in MATLAB file format. In plain text file format, it is given as a separate .txt file under the input folder, which has four lines and each line consists of four variables separated by spaces to form the matrix. Examples can be seen on Figure 3.10.



(a) (b)

Figure 3.10: Two forms of MNI transformation matrix input. Figure 3.10a shows the MAT-File format version as a variable in a .mat file. Figure 3.10b is the same data in plain text file format.

3.2.6 Brain Parcellation Labels (External Indexing)

Rapid improvements in hardware technologies (GPUs, clusters etc.) allow many scientists to run many machine learning algorithms from k-means to deep neural networks on brain networks. These algorithms allow them to investigate the relation between the functional regions and matching between functional regions with anatomical regions in order to diagnosing disorders [34, 52], hypothesis validation [54] and classifying cognitive states [43, 21, 73, 53]. Therefore, visualization of the outcomes of any clustering algorithm or classifier with the relations between the regions is a crucial task.

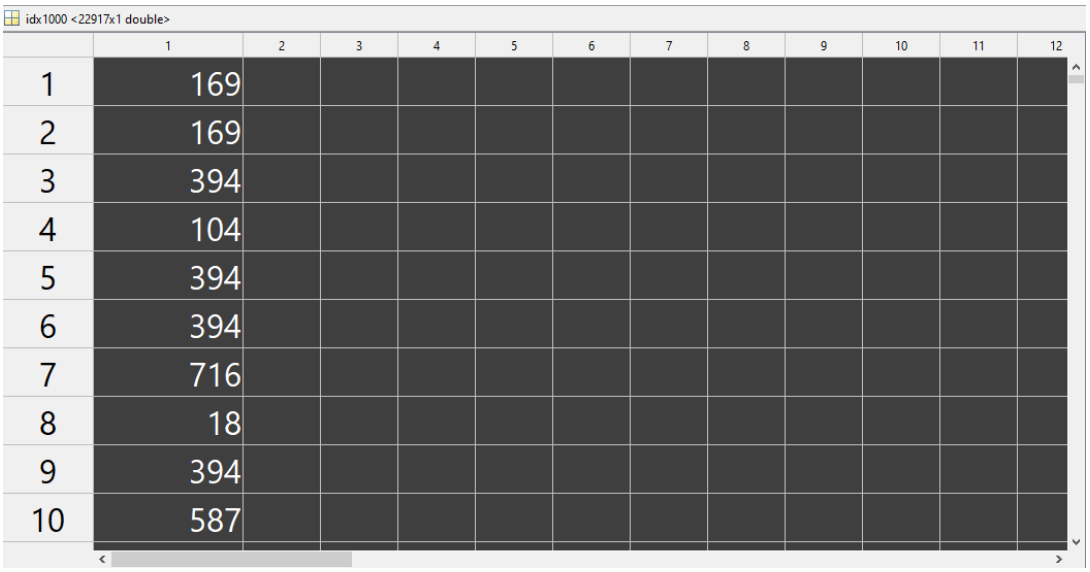


Figure 3.11: An example functional label variable in MATLAB. This file holds the output of the algorithm proposed by Mogultay et al. [43] in which 10,000 voxels are divided into 1000 groups (clusters). The matching between voxels and labels are done by using their line numbers as we did on other variables.

In order to visualize the output of these algorithms, the user should provide a separate .mat file to the system. This information can only be loaded in MATLAB file format. Therefore, even if the user has uploaded the brain data in plain text file format, the label information should be prepared in MATLAB file format.

The related .mat file should include one or more variables that may contain different

types of labels of each voxel in a column. Therefore, the format of the variable should be an $N \times 1$ matrix, where N is the number of voxels in the voxel position variable. An example file and its structure is shown in Figure 3.11.

3.3 Pre-Processing Steps of CEREBRA

As mentioned in the Background chapter (Section 2.3.1), the data goes through some pre-processing stages (normalization, correction etc.) before it's fed to CEREBRA. In some cases, this preprocessing is enough for generating a visualizable data. In most cases, further processing operations on fMRI data are necessary in order to make the analysis on a visual platform. In this section, we will have a look at how we organize the data in CEREBRA and the pre-processing operations from re-centering the data to assigning anatomical regions.

3.3.1 Packet Structure

Like all other tools and programs, CEREBRA also needs to hold the information in a systematic data structure after reading the input. In order to accomplish this task, we have designed a class called *Packet*, that approaches to the brain in both structural and functional perspectives. The structure of the *Packet* class can be seen in the diagram given in Figure 3.12. As can be seen in the diagram, *Packet* encapsulates two other classes namely, *VoxelPositions* and *Voxel*.

VoxelPositions is a simple class definition that holds three variables indicating the Euclidean coordinates of a point, in our case a voxel. This class is used in *Voxel* class in order to specify each voxel's position in 3D Euclidean space.

Voxel is a class that imitates a neuron in the real brain structure. Due to the low-resolution images obtained from fMRI, *Voxel* class is actually a low-resolution approximation to a neuron (it consists of more than thousand neurons). However, since a voxel is the smallest element that we can get from the whole brain data, we can approach it as a neuron in this work. Therefore, a voxel contains the following fields

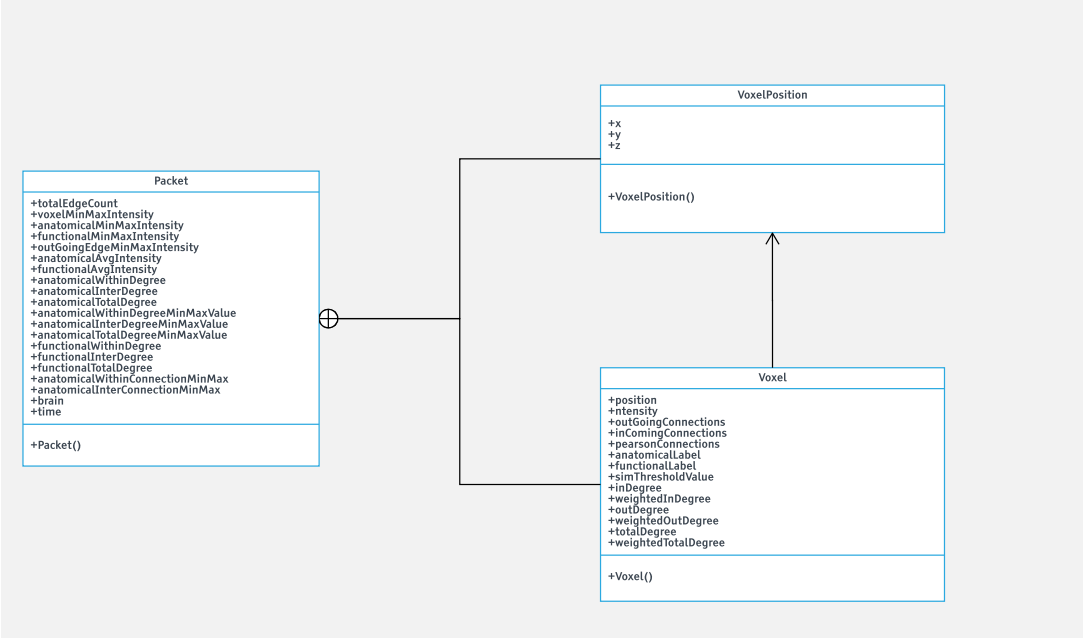


Figure 3.12: The *Packet* class diagram. *VoxelPositions* and *Voxel* classes are defined within the main *Packet* class.

to reflect some properties of the brain:

- **position:** This field reflects the position of the voxel in 3D Euclidean space by using the *VoxelPosition* class.
- **intensity:** This field represents BOLD signal fluctuations of the voxel during the experiment. This variable is a float vector in order to hold the time information.
- **outGoingConnections:** This field holds the connections emerging from the voxel through the others. In this field both the i.d. of the neighbor and connection's weight information are stored. The weights are subject to change in time, therefore, they are stored in a float vector. The matching between weights and the neighbor i.d. is done via pairing.
- **inComingConnections:** This field holds the edge existence information coming through the voxel. Since the weight information of those edges are stored in the *outGoingConnections* fields of the neighboring voxels, this field only holds

the i.d.'s of the neighboring voxels. As long as we know the existence of an incoming connection to voxel i from voxel j we can check it's value from the *outGoingConnections* map of the voxel j. This enables us to save memory and avoid from storing the same data twice.

- **pearsonConnections:** This field holds the Pearson correlation coefficients computed by CEREBRA (Pearson correlation coefficient calculator). The structure is the same with *outGoingConnections*. This field is added in order not to lost loaded connection information, when the user calculates the Pearson correlation coefficients through the tool.

- **anatomicalLabel:** All anatomical labels of the voxel are kept in this field. This field has six slots for each anatomical atlases (Hemisphere, Lobe, Type, Label Extended, Broadmann and Automated Anatomical Labeling) assigned by CEREBRA.

- **functionalLabel:** This field is used when the user loads external indexing (label) information to CEREBRA. This field is filled according to the voxel's external labels fetched from the loaded label file.

- **simThresholdValue:** This field holds the unique threshold value computed for the voxel by the local edge sparsifier method (details on Section 4.6).

- **inDegree:** Number of incoming connections for the voxel is held in this field.

- **weightedInDegree:** This field keeps the sum of weights of the incoming connections.

- **outDegree:** This field holds the number of outgoing connections of the voxel.

- **weightedOutDegree:** Sum of weights of outgoing connections is kept in this field.

- **totalDegree:** Number of all connections belonging to the voxel.

- **weightedTotalDegree:** Sum of weights of all connections belonging to the voxel.

Besides from these two classes, the *Packet* class has some other fields to hold information well-ordered. These fields are listed with their descriptions as follows:

- **brain:** This field is a vector of *Voxel*s. As neurons form the brain in the actual world, in our case the *brain* is formed by a low-resolution representation of the neurons, voxels, in order to simulate the real one.
- **time:** This field holds the number of time instances in order to animate the cognitive experiment on display.
- **totalEdgeCount:** This field holds the total number of edges in the loaded data.
- **voxelMinMaxIntensity:** Global minimum and maximum intensity values are kept in this field. Even if the time information is available, it only holds only one pair of minimum and maximum values observed during the whole experiment. In other words, we do not compute the minimum and maximum values of each time instance separately.
- **anatomicalAvgIntensity:** This field holds the average intensity values for each anatomical region inside each anatomical atlases.
- **functionalAvgIntensity:** This fields holds the average intensity values for each region, provided externally by the user.
- **anatomicalMinMaxIntensity:** Global minimum and maximum values for the anatomical average intensities are kept in this field.
- **functionalMinMaxIntensity:** Global minimum and maximum values for the external labels, functional regions, are kept in this field.
- **outGoingEdgeMinMaxIntensity:** This field holds the minimum and maximum weight values for the edges observed during the whole experiment.
- **anatomicalWithinDegree:** This field holds the number of self-connections for each anatomical region inside each anatomical atlases.
- **anatomicalInterDegree:** This field holds the number of outgoing-connections for each anatomical region inside each anatomical atlases.

- **anatomicalTotalDegree:** This field holds the number of all connections for each anatomical region inside each anatomical atlases.
- **anatomicalWithinDegreeMinMaxValue:** Global minimum and maximum values for anatomical within degrees among all regions are kept in this field. For each anatomical atlases, there exists a unique pair of minimum and maximum values in this field.
- **anatomicalInterDegreeMinMaxValue:** Global minimum and maximum values for anatomical inter degrees among all regions are kept in this field. For each anatomical atlases, there exists a unique pair of minimum and maximum values in this field.
- **anatomicalTotalDegreeMinMaxValue:** Global minimum and maximum values for anatomical total degrees among all regions are kept in this field. For each anatomical atlases, there exists a unique pair of minimum and maximum values in this field.
- **functionalWithinDegree:** This field holds the number of self-connections of each label, loaded externally by the user.
- **functionalInterDegree:** This field holds the number of outgoing-connections of each label, loaded externally by the user.
- **functionalTotalDegree:** This field holds the number of all connections for each label, loaded externally by the user.
- **anatomicalWithinConnectionMinMax:** This field holds the global minimum and maximum edge weights values for anatomical regions' self-connections. For each atlases, there is a unique minimum and maximum value in this field.
- **anatomicalInterConnectionMinMax:** his field holds the global minimum and maximum edge weights values for anatomical regions' outgoing connections. For each atlases, there is a unique minimum and maximum value in this field.

Each of these fields is used to form and visualize the brain on the screen in the correct shape and with correct coloring. This class can be extended only by editing the header

and source files belonging to the *Packet* class without needing to change anything on the main project. The *Packet* class is used as an external library on the main Qt project in order to ease the developers' work.

3.3.2 Normalization and Correction of fMRI data

In order to offer a better visualization on the display and enable user interactions on the 3D graph, the data has to go under some pre-processing operations. These operations take two main steps, namely, MNI transformation and brain position correction (re-centering).

As stated in Section 2.3.1, human brain atlases are used to map the location of brain structures independent from individual differences in size and overall shape of the brain. Therefore, the brain normalized to a certain standard for multi-subject analysis to match and compare brain regions and their intensities/relations. As anatomical labels are, also, defined for these standard templates, we need to perform this normalization before assigning the labels. There are two main standard maps available for human brain, namely, Talairach and MNI coordinates. In this thesis, we use MNI space to normalize varying sizes and shapes of the human brain. In order to transform a brain matrix from subject's space to MNI coordinate space, the subject coordinates should be multiplied with MNI transformation matrix as shown in Eq. 3.1. The first matrix in the equation holds $s_i = (x_i, y_i, z_i)$'s of each voxel $i = 1, \ldots, N$, where N is the number of voxels. The values, $v_{1,4}$, $v_{2,4}$ and $v_{3,4}$, in the second matrix (MNI transformation matrix) stand for translate the positions, whereas the diagonal values scale the graph according to MNI coordinate space. The transformation matrix is calculated by computing the linear transformation between the individual's space and the MNI template. The calculation can be operated by third party tools, e.g. SPM Toolbox [5]. The user may upload the MNI transformation matrix as an input to CEREBRA, if available, in order to transform the graph into MNI coordinate space. If this matrix is not available, the uploaded brain matrix could be multiplied with the

default transformation matrix in Eq. 3.2 or left as it is.

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & 1 \end{bmatrix} \times \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} \\ v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} \\ v_{3,1} & v_{3,2} & v_{3,3} & v_{3,4} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{\top} = \begin{bmatrix} x'_1 & y'_1 & z'_1 & 1 \\ x'_2 & y'_2 & z'_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x'_N & y'_N & z'_N & 1 \end{bmatrix} \tag{3.1}$$

$$MNIT_{default} = \begin{bmatrix} -4 & 0 & 0 & 82 \\ 0 & 4 & 0 & -116 \\ 0 & 0 & 4 & -54 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{\top} \tag{3.2}$$

The transformation is performed by using Eigen library functions in order to calculate matrix multiplications faster [2]. The performance graph of Eigen library can be seen on Figure 3.13. Note that the higher values in MFLOPS are better. MFLOPS means millions of (effective) arithmetic operations per second. The reason why the values are typically low for small sizes is that in this benchmark, we deal with dynamic-size matrices which are relatively inefficient for small sizes. The reason why some libraries/benchmarks show a decline for large sizes is that for such large matrices issues of CPU cache friendliness becomes predominant. However, in our case, the minimum matrix size is $40{,}000$ which exceed that separation limit.

There are many preprocessing algorithms for data correction available and used by the researchers. Although each of these algorithms preserves the brain structure and integrity, they may have different assumptions and applications on voxel positions. For example, an algorithm may re-position the brain such that each position value should be greater than zero. In such cases, we have to re-center the brain by subtracting voxel position mean value from all dimensions. Therefore, each voxel is shifted by

$$s'_i = s_i - \mu_s, \tag{3.3}$$

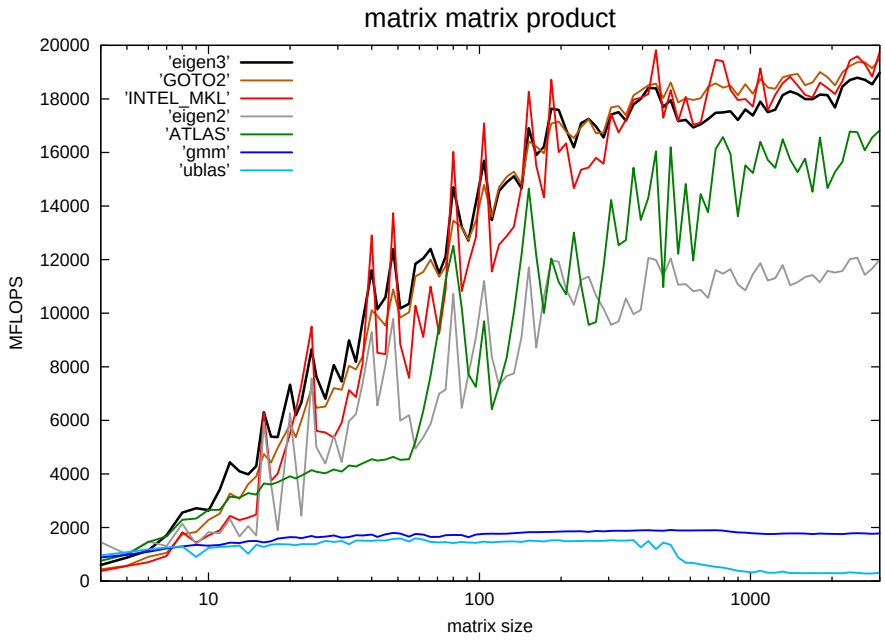


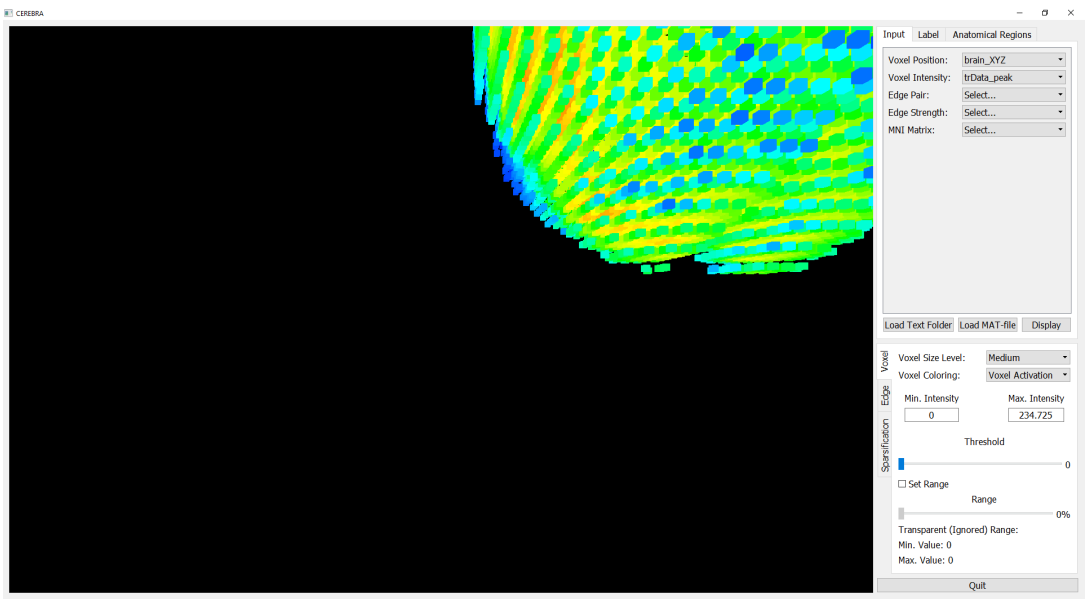
Figure 3.13: Matrix - Matrix multiplication performance comparison graph. Test was carried out using Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz (x86_64) and C++ (SUSE Linux) 4.5.0 20100604 [gcc-4_5-branch revision 160292] as the compiler. The figure is taken from [2].

where $s_i = (x_i, y_i, z_i)$ represents the position of the voxel i in 3D Euclidean space (as mentioned in Eq. 2.4) and μ_s is calculated as follows:

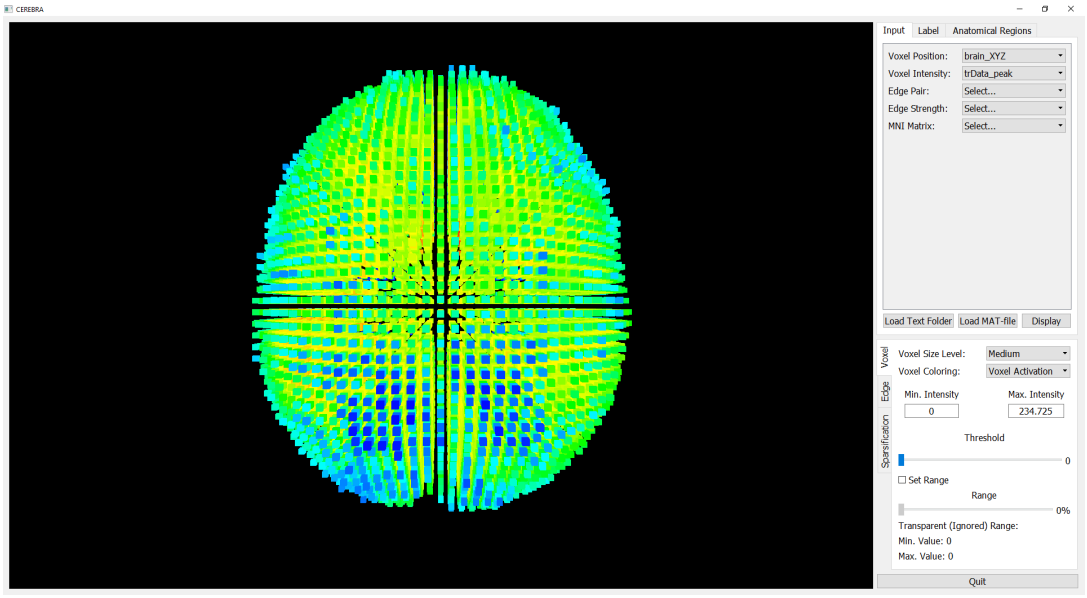
$$\mu_s = (\frac{\sum_{i=1}^{N} x_i}{N}, \frac{\sum_{i=1}^{N} y_i}{N}, \frac{\sum_{i=1}^{N} z_i}{N}), \tag{3.4}$$

where N is the number of voxels.

An illustration of the difference between visualized graphs with and without this correction is given in Figure 3.14. Since user interaction functionalities, such as zooming and rotation, assumes that the data is centered at the point $(0, 0, 0)$ on the display, these functionalities would not work properly on a graph placed as shown in Figure 3.14a.



(a)



(b)

Figure 3.14: Effect of the voxel position correction on the brain graph. These images are taken just after the data is displayed without any translation, rotation, or zoom operations are performed on the graph. Figure 3.14a illustrates an example, where the re-centering feature is blocked. Figure 3.14b is the same graph re-centered by CEREBRA where the center of the graph is matched with the center of the display grid.

3.3.3 Assigning Anatomical Regions

Sometimes scientists change the scale of their analysis level from voxel analysis to regional analysis. In this scenario, they examine the relationship between anatomical or functional regions to explore the location of certain cognitive processes or diseases. Although the functional regions are subject to change with an employed algorithm (that outputs the external indexes) and its parameters, anatomical atlases do not change by any parameter or brain size. Therefore, we are able to assign all anatomical regions for specific atlases just after the user loads the data.

Anatomical region assignment enables users to switch voxel color codes from activity record to anatomical mapping or observe only the selected regions. The information is preloaded to the system for six anatomical atlases in MNI coordinate space. However, the user should provide the MNI transformation matrix mentioned in the previous subsection or load the voxel positions directly in MNI coordinate space, in order to enable CEREBRA to assign anatomical regions. Otherwise, the program will assume the graph is in MNI coordinate space and assigns wrong labels to voxels.

The pre-loaded anatomical label file (.txt) is prepared by matching all possible MNI voxel positions, taken from Marsbar [13] and the map used in xjView toolbox [6], with six brain atlases. Each line consists of nine numbers divided by a sharp (#) sign in which the first three numbers represent the MNI space coordinates and the following numbers stand for region labels for Hemisphere, Lobe, Type, Label Extended, Broadmann and Automated Anatomical Labeling (AAL) V4 atlases respectively (Figure 3.15). Since the anatomical label file is in a simple plain text format, it could be extended and updated by anyone without any programming languages background and without needing of altering the structure of the tool or building it from scratch.

As the user loads the input, the tool employs this plain text file in the background and prepares a map using the position and label information. Then, the algorithm searches every voxel with their positions in this map and if that position exists in the map, the following label information is assigned to that voxel. If the search returns null, then the voxel takes 0 for all label information which means it gets the label

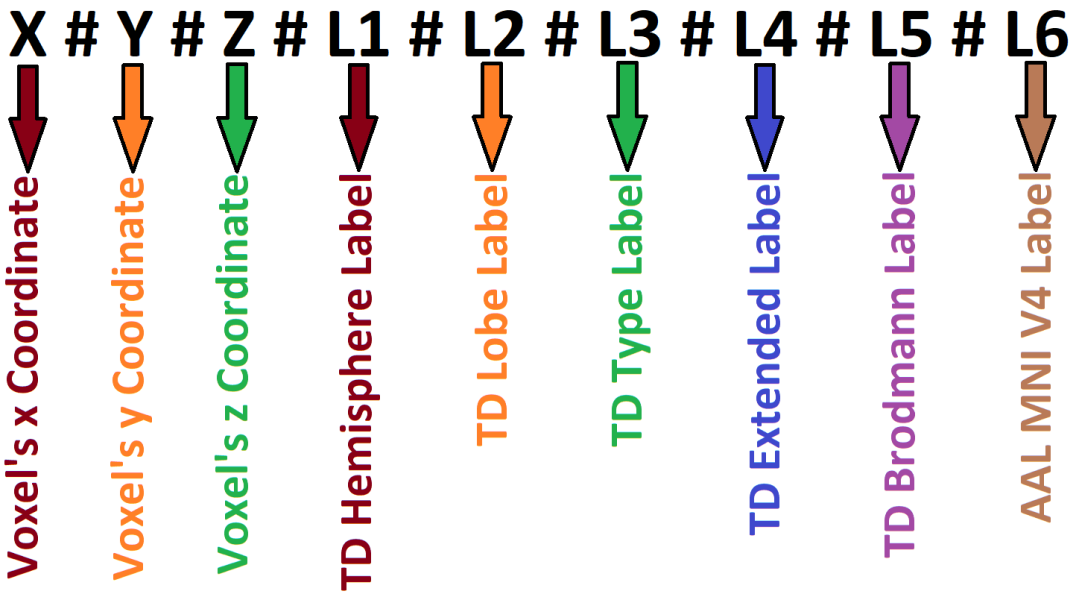


Figure 3.15: Anatomical label format. The first three variables are the Euclidean space coordinates and the rest are the anatomical labels. Label values are integer whereas the coordinates could be double. The sharp characters between the values are separators.

unknown. Please note that, even if the voxel is found on the position-label map, it may get the *unknown* label for some anatomical mappings as these maps have also some undefined regions.

The mapping between colors and regions are defined in the "anatomicalinformation.h" file in the main project. This header also contains the names of anatomical mappings and the regions lie within them. If the user needs to change the color, opaqueness or name of a region, then the user should modify this header file.

3.4 Chapter Summary

In this chapter, we have explained how we structured the input fMRI data and how we hold it in CEREBRA to process it further. In order to ease the workload on neuroscientists and many other possible users, we have employed two common file types as the acceptable input file format, namely, MATLAB file format (.mat) and plain

text format (.txt). Although two file formats are supported, they have key differences when it comes to the input structure. The entire information should be loaded in a single .mat file in MATLAB file format, where each input parameter is a MATLAB variable whereas each input parameter is encapsulated by an individual .txt which are stored under the same folder in plain text file format. For both file types, there could be more than one parameters for the same input field since the user is able to choose the parameters for displaying from the tool itself. That way the user may observe many experiments on the same data without needing to reload "mostly" the same data over and over again. The tool, also, allows the users to display label information on the loaded data. However, the label information should be given in a separate .mat file and loaded after voxel position information. Label information is only supported in MATLAB file format and as in the other .mat file specifications, this .mat file may encapsulate more than one label variable in it. The label information can be changed through the tool without loading a new file for each change. The information is stored in the *Packet* class, in which the brain is defined as a list of *Voxel*s. The *Voxel* class contains the primary information about one neuronal unit (in our case we call this unit voxel) such as position, intensity, list of connected neuronal units and weight of these connections. The other variables lies in the *Packet* class hold some global information about the brain, such as functional and anatomical region intensities and their connections.

Although the data has passed many pre-processing steps before it takes the final form, we need to further process it to enhance the displayed graph. There are three basic pre-processing operations done by CEREBRA, which take stage between the time interval just after the user presses the "Display" button and the graph is drawn on the display. The first one is transforming data into MNI coordinate space. The positions of voxels are multiplied by the MNI transformation matrix, if available. If no transformation matrix is loaded to the system by the user, then the tool multiplies the position matrix with a default transformation matrix or left it as it is. After transformation operation is done, anatomical labels are assigned to voxels by using the anatomical-position map prepared for MNI coordinate space with various resolutions. Each voxel takes six labels from each of the following atlases: Hemisphere, Lobe, Type, Ex-

tended, Broadmann and AAL MNI V4. In the final stage of pre-processing, the data is spatially corrected to shift its center to the origin in all dimensions. This operation enables users to perform rotation and zoom in and out operations on the displayed data.

CHAPTER 4

GRAPH SPARSIFICATION AND SIMPLIFICATION METHODS OF CEREBRA FOR INFORMATION EXTRACTION

CEREBRA is designed for a wide range of researchers who are interested in the visual analysis of human brain. For this purpose, we keep the tool simple, easy-to-use and ordered. We have used QT UI Development Framework [4] for building the user interface of CEREBRA. QT does not only ease design process of user interface (with drag and drop UI development) but also enables us to achieve a simple design. The resulting user interface is also stored in XML format, which could also be edited from a text editor for future updates.

The user interface of CEREBRA is divided into two main segments, namely, display and option panels. The first one is where the brain graph is displayed and some affine transformations are performed on the graph. The option panel is divided into six tabs which are grouped into two categories; non-processing and processing tabs. Non-processing tabs are where the user loads input, switches between regional view and intensity view etc. The processing group is where the actual graph structure is altered and processed. All essential parts and functions of CEREBRA are illustrated in Figure 4.1.

In the following subsections, the functions shown in the diagram are explained and exemplified.

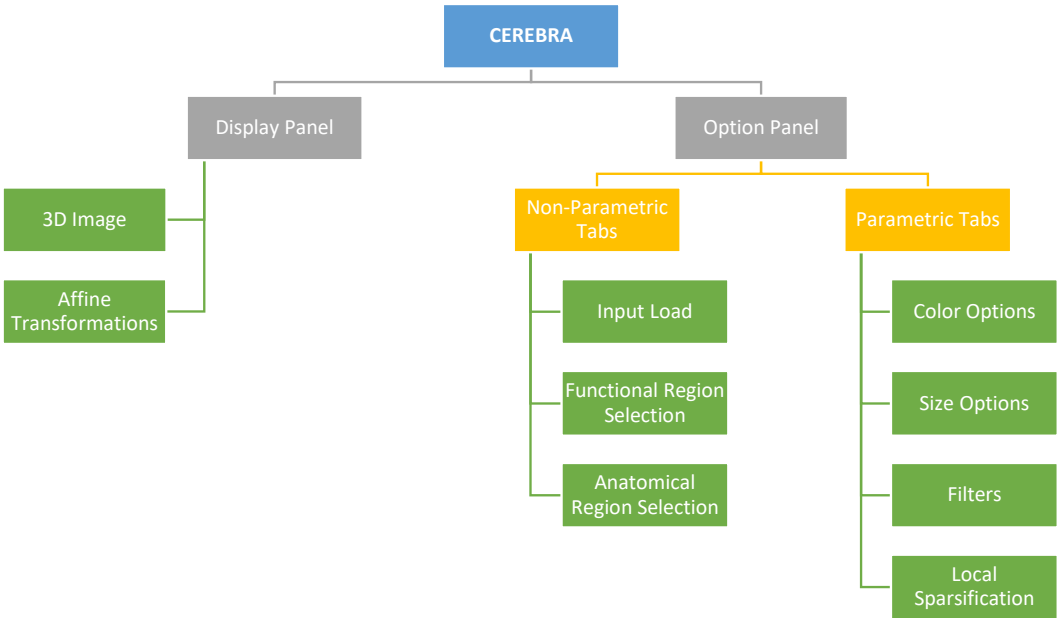


Figure 4.1: Basic parts and functions of CEREBRA. Although we have many functions and parameters to run the tool, the figure shows the key points we have explained in this thesis. Gray and yellow boxes indicate the divisions of the tool whereas the green ones stand for functionalities.

4.1 Coloring

Voxel and edge color codes can be managed to reflect specific aspects of the data. However, they share some common things, such as, color scheme, algorithms to manage color codes and variables in the shading language. In this section, we give the details of common things shared by each color mapping in details and continue with the specific details of each color mapping in the following subsections.

In order to indicate voxel intensity, voxel degree or edge weight information, we need to pick a color scheme first. In this work, we have employed the Hot-to-Cold color scheme in order to map intensity/degree/weight values to colors. The color spectrum is given in Figure 4.2. We have picked this color scheme because it is quite similar to the Jet color scheme, which is the default option in MATLAB and used in BrainNet Viewer, one of the popular tools in this research area. As can be seen in the Figure 4.3,

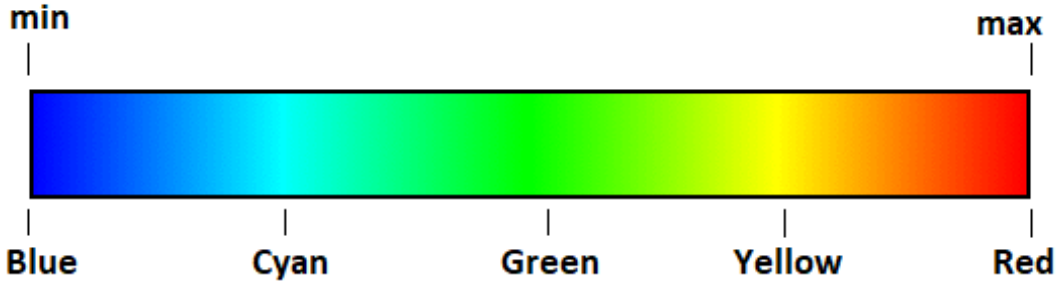
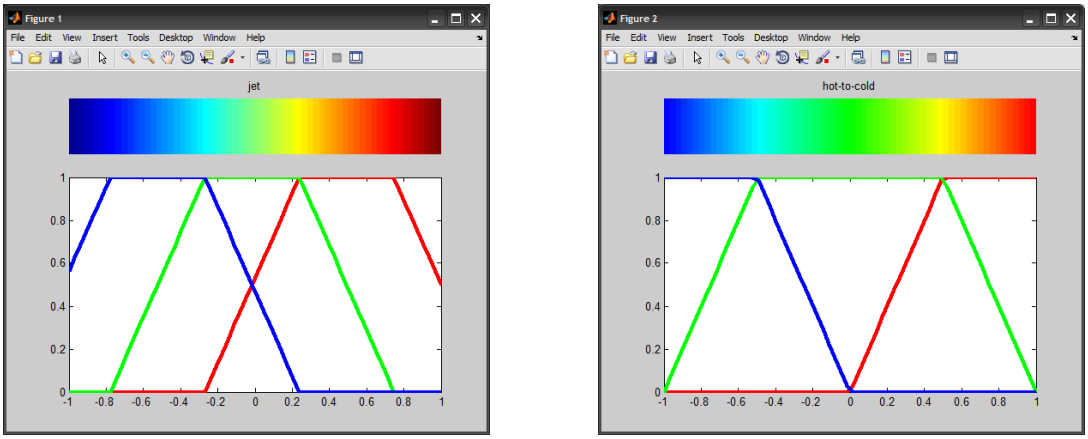


Figure 4.2: Hot-to-Cold color spectrum. The spectrum starts with color blue which indicates the minimum value in the given range and continues until the red which indicates the highest value in the given range. The spectrum has three additional breakpoints which are cyan, green and yellow. Green is used to indicate mid range values.



(a) Jet Color Scheme

(b) Hot-to-Cold Color Scheme

Figure 4.3: Jet and Hot-to-Cold color schemes and their plots. Notice the difference between the mid values on both plots. In the Hot-to-Cold color scheme, green spans more mid ranged values than it does in the Jet color scheme. This allows researchers to discriminate lowest and highest values from mid ranged values more easily. The image is taken from [1].

both color schemes start from blue, which indicates the lowest value and ends with red, which reflects the highest value in the given data. They also share the same colors on their ramps. However, in Jet Color scheme, the discrimination between low and mid values is quite low compared with the Hot-to-Cold color scheme. This

is mainly because the Jet color mapping does not have an emphasis on mid values which should be colored as green. Blue and red colors cover too much space on the spectrum, which makes discrimination of mid values to highest or the lowest ones. On the other hand, Hot-to-Cold mapping gives more space to green, which enables us to limit the boundaries of red and blue colors. Consequently, highest and lowest values in the data become more distinct. In addition, in CEREBRA, if the voxel intensity, voxel degree or edge weight value is 0, they are considered to have no contribution. These voxels and edges are drawn as ghosts, colorless, in order not to confuse the user. Hot-to-Cold color scheme calculation for an input value x and known global minimum and maximum values, x_{min} and x_{max}, is given in Eq. 4.1.

$$[r,\, g,\, b] = \begin{cases} [0.0\,,\, \dfrac{4 \times (x - x_{min})}{d}\,,\, 1.0] & x < (x_{min} + 0.25 \times d) \\ [0.0\,,\, 1.0\,,\, 1 + \dfrac{4 \times (x_{min} + 0.25 \times d - x)}{d}] & x < (x_{min} + 0.50 \times d) \\ [\dfrac{4 \times (x - x_{min} + 0.50 \times d)}{d}\,,\, 1.0\,,\, 0.0] & x < (x_{min} + 0.75 \times d) \\ [1.0\,,\, 1 + \dfrac{4 \times (x_{min} + 0.75 \times d - x)}{d}\,,\, 0.0] & otherwise, \end{cases} \tag{4.1}$$

where the array contains red (r), green (g) and blue (b) color codes of a voxel with intensity/degree value(or an edge with weight value) x respectively and the distance between global minimum and maximum values d is calculated as:

$$d = x_{max} - x_{min}. \tag{4.2}$$

In Figure 4.4, the four breakpoints in the Eq. 4.1 is shown on the color spectrum. In addition, a 3D illustration of the color spectrum in RGB space is given in Figure 4.5, where we start from blue and walk through red, passing by cyan, green and yellow. In addition, please note that this color mapping calculation normalizes the given value into $[0, 1]$ interval. However, this normalization does not change the actual voxel intensity/degree or edge weight value. It is performed only to obtain a color code from the spectrum.

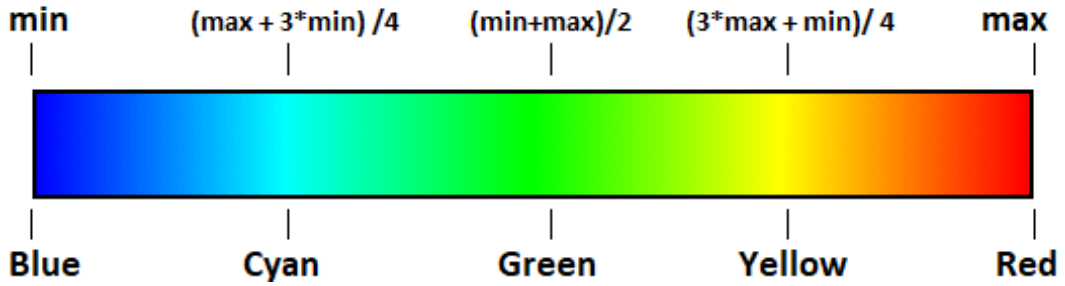


Figure 4.4: The breakpoints on Eq. 4.1 is marked on the Hot-to-Cold color spectrum.

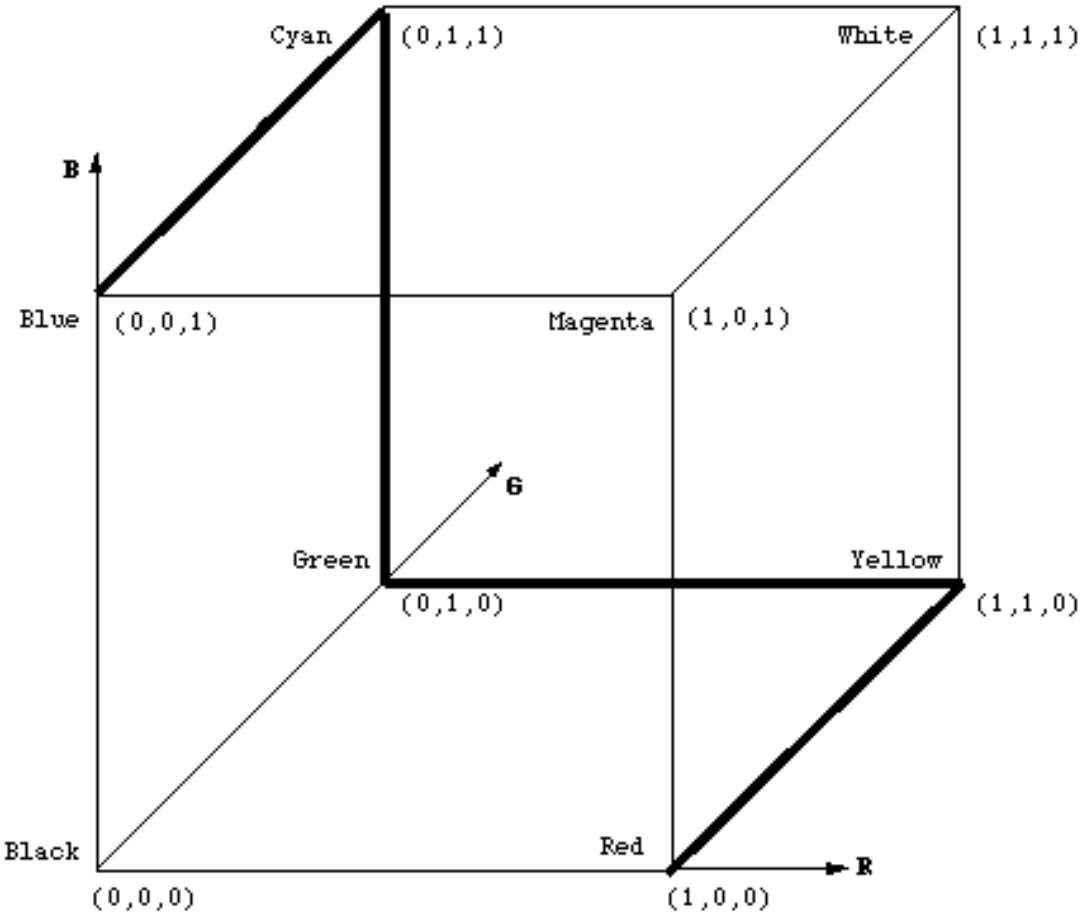


Figure 4.5: The path of the color spectrum in three-dimensional RGB space. As it is seen, the spectrum starts from blue and walk through red (passing by cyan, green and yellow) interpolating the values on this path. The image is taken from [1].

The calculation is done by an OpenGLSL (OpenGL Shading Language) [62] function, which employs the graphics processing unit (GPU) power to accelerate the compu-

tation by parallelization. In other words, each voxels' color codes are calculated and assigned by the shading language in parallel to make calculations faster. In order to transfer the input values (intensity, degree or weight) to shading language, we used a texture-map. The trick is, we fill the texture-map with the intensity, degree or weight values as if they are the actual texture, but, in the shading language itself, we use these values to compute the actual color of the voxels and edges. For each color code generation, shading function fetches a value from the texture-map, indexed according to the processed voxel or edge, and do the calculation. For each color mapping, mentioned under this section, we fill the texture-map accordingly to display selected modality of the data.

In the rest of this section, we have detailed the color options and their implementation details. Since we have different approaches for voxels, edges, and regions on coloring and animation of them we have divided this section into four subsections.

4.1.1 Voxel Coloring

Voxel color codes may reflect much information about the brain and its operation. This information ranges from intensity to anatomical mappings or from voxel based connection density to region based connection density. In this subsection, we have detailed each possible voxel color mapping types and our approach to reflect them on the display correctly.

4.1.1.1 Voxel Intensity Mapping

Recall that the attribute of each voxel is a time series intensity $v(t, s_i)$ for $t = 1, ..., T$, where T represents the total number of time samples and s_i represents a three dimensional coordinate of the voxel i. This intensity is related with Blood Oxygenation Level Dependent (BOLD) signal on fMRI experiments. BOLD signal refers to the change in oxygen saturation of the hemoglobin molecules causes small alterations in the local MR signal. When a neuronal unit activates, it needs more oxygen which causes fluctuations on BOLD signal [48]. The resolution of fMRI machines allows

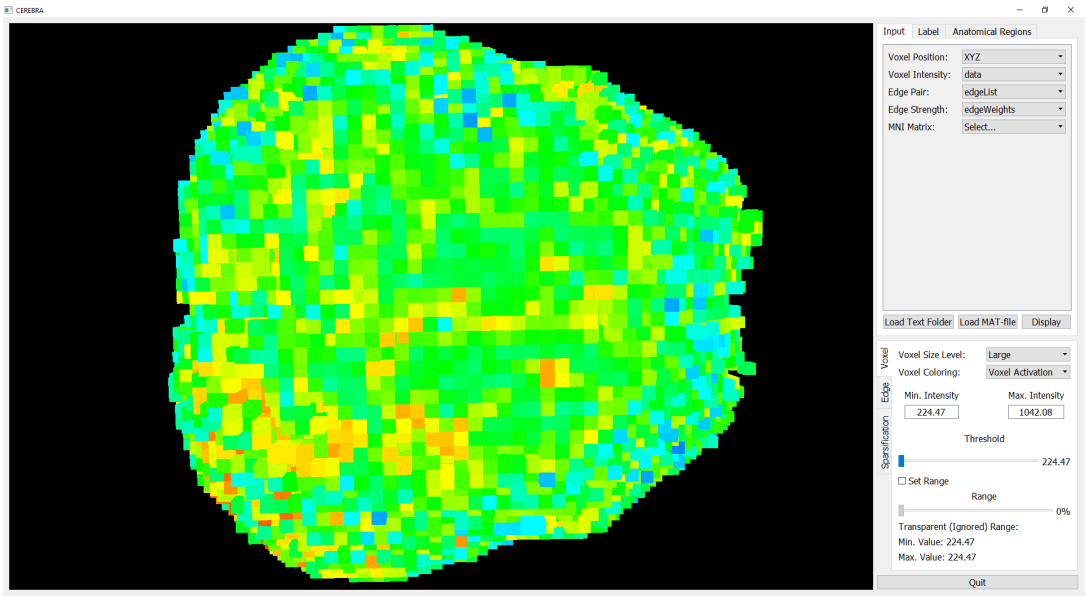


Figure 4.6: An example voxel intensity visualization on CEREBRA. Voxel sizes are magnified in order to observe intensities easily. Voxel intensity values decrease from red color to blue color. Green color represents the mid-valued intensities.

us to track neuronal units that consist of thousands of neurons, called voxels. Each voxel reflects the average intensity, which resides the intensity of neurons in that voxel. Therefore, we call this mapping between colors and low-resolution BOLD signals as voxel intensity mapping.

We have employed the Hot-to-Cold color scheme as mentioned earlier in this section. The only alteration on the color scheme is that the algorithm displays the 0 intensity value, regardless of the intensity interval, as colorless, since zero valued intensity indicates no activation.

Voxel coloring is handled and implemented in OpenGLSL. Therefore, the texture-map, which handles the communication between GPU and host (CPU), is filled with the values of $v(t, s_i)$. The steps carried out by the shading language are detailed in Algorithm 4.1.

An example to voxel intensity colorization is given in Figure 4.6. The graph in the figure is obtained from Onal et al.'s work [51]. The voxel intensities are recorded during a visual object recognition experiment. In addition, Figure 4.7 and Figure 4.8

illustrate the same graph with thresholded intensity values and rotated in order to observe the most active voxels.

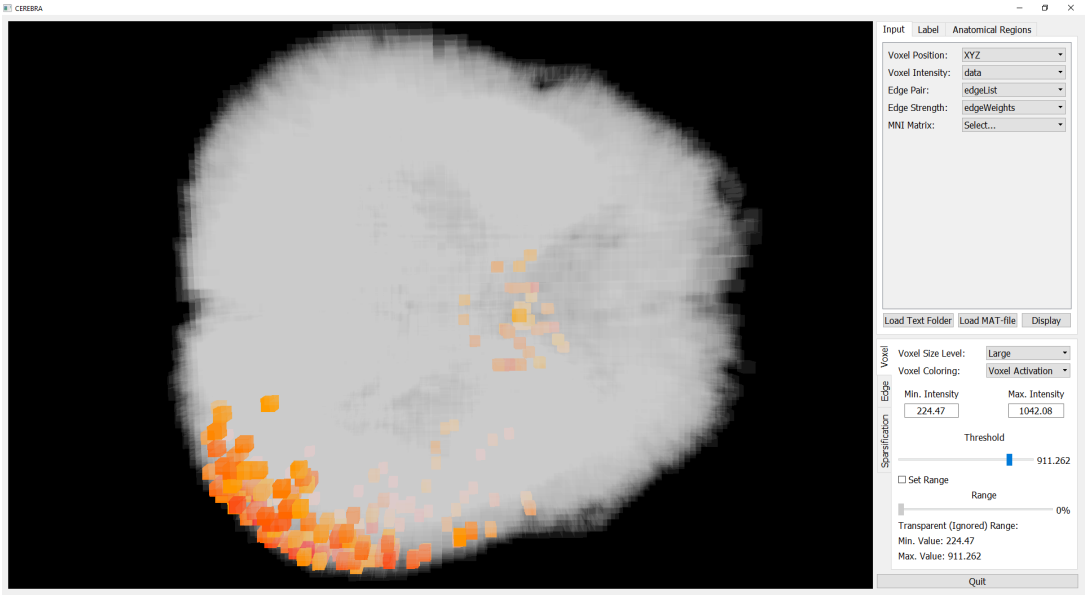


Figure 4.7: Thresholded version of the Figure 4.6. We allowed the voxels with intensity value above 911.262.

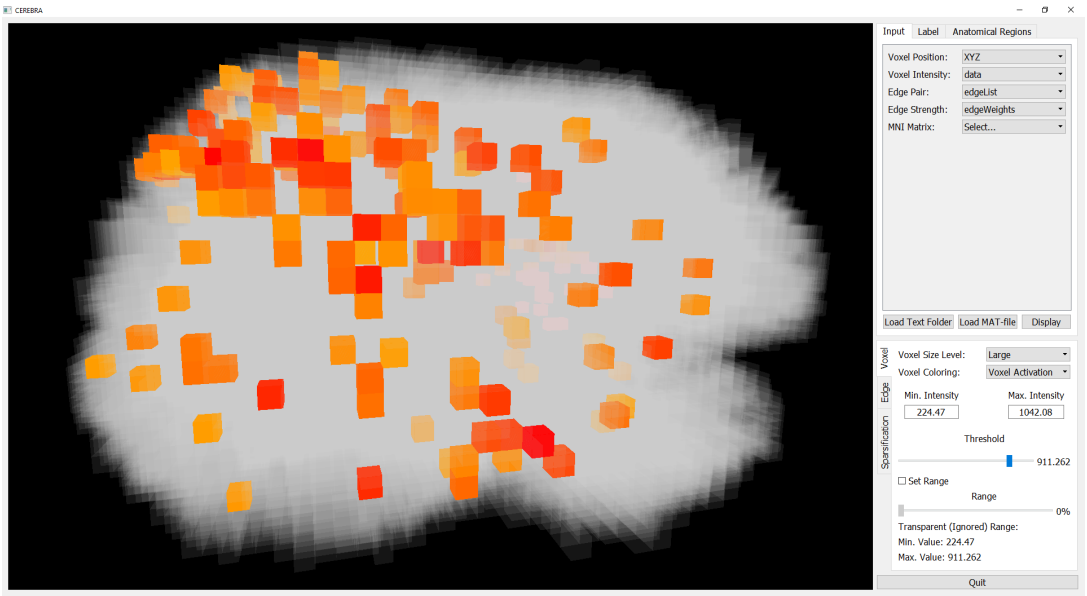


Figure 4.8: Rotated version of the Figure 4.7.

Algorithm 4.1 Voxel Coloring Steps in OpenGLSL

1: $intensity := textMap[currentVoxel + textureOffSet]$
2: $minValue :=$ The lowest voxel intensity value in the loaded data
3: $maxValue :=$ The highest voxel intensity value in the loaded data
4: $r := 0.0$
5: $rMin := 0.0$
6: $rMax := 1.0$
7: $minMaxDistance := 0.0$
8: $red, green, blue := 0$
9: $r =$ Normalize$(intensity, minValue, maxValue)$
10: **if** $minValue < 0$ **then**
11: $\quad rMax := maxValue/(maxValue - minValue)$
12: $\quad rMin := minValue/(maxValue - minValue)$
13: $\quad minMaxDistance := rMax - rMin$
14: **else**
15: $\quad minMaxDistane := 1.0$
16: **end if**
17: **if** $r < (rMin + 0.25 \times minMaxDistance)$ **then**
18: $\quad red := 0$
19: $\quad green := 4 \times (r - rMin)/minMaxDistance$
20: **else if** $r < (rMin + 0.50 \times minMaxDistance)$ **then**
21: $\quad red := 0$
22: $\quad blue := 1 + 4 \times (rMin + 0.25 \times valueDistance - r)/valueDistance$
23: **else if** $r < (rMin + 0.75 \times minMaxDistance)$ **then**
24: $\quad blue := 0$
25: $\quad red := 4 \times (r - rMin - 0.5 \times valueDistance)/valueDistance$
26: **else**
27: $\quad blue := 0$
28: $\quad green := 1 + 4 \times (rMin + 0.75 \times valueDistance - r)/valueDistance$
29: **end if**
30: Return $vec4(red, green, blue, 1.0)$

4.1.1.2 Voxel In-Degree and Weighted In-Degree Mapping

This mapping colors the voxels according to their incoming edge numbers (in-degree option) or the sum of incoming edge weights (weighted in-degree option). We have used the same color spectrum with same modifications as we did on voxel intensities. Therefore, when the in-degree option is selected, if a voxel is connected by a large number of voxels, its color converges to pure red. Consequently, if a voxel is connected by less number of voxels, even if it connects to the others, its color turns to pure blue. If the voxel has no incoming connections, then this voxel becomes a ghost (colorless).

The in-degree value, id_i, of voxel i is calculated as follows:

$$id_i(t) = \Sigma_{k=1}^{N} 1_{A(t,\, s_k,\, s_i) \neq 0}, \tag{4.3}$$

where, k indicates the other voxels in the graph and $A(t,\, s_k,\, s_i)$ is the edge weight time-series of the edge emerges from voxel k and goes through to voxel i. In this equation, we add 1 to id_i if the weight between voxel k and voxel k does not equal to zero. In weighted in-degree option we directly sum all weight values of the edges coming through voxel i as follows:

$$wid_i(t) = \Sigma_{k=1}^{N} A(t,\, s_k,\, s_i), \tag{4.4}$$

where wid_i holds the weighted in-degree value for voxel i.

If the weighted in-degree option is selected, the color turns red when a voxel's total weights of incoming neighbors are higher than the others. In other words, even if the voxel is connected by just one voxel and this edge weight is greater than most of the weighted sum of other voxels' incoming connections, then this voxel colored by a reddish color. If the result of this sum equals to zero, then the voxel becomes a ghost.

The coloring method is the same with illustrated in Algorithm 4.1, however, this time we have filled the texture-map with voxel in-degree values. In other words, instead of

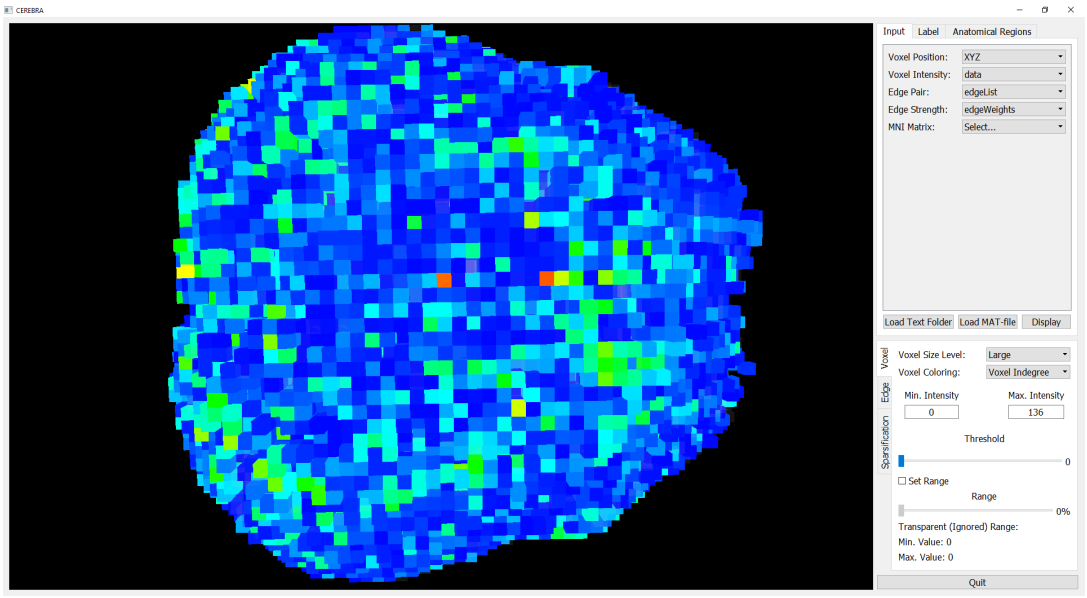


Figure 4.9: An example voxel in-degree visualization on CEREBRA. The connections are hidden to observe voxels clearly. Voxel in-degree values decrease from red to blue color.

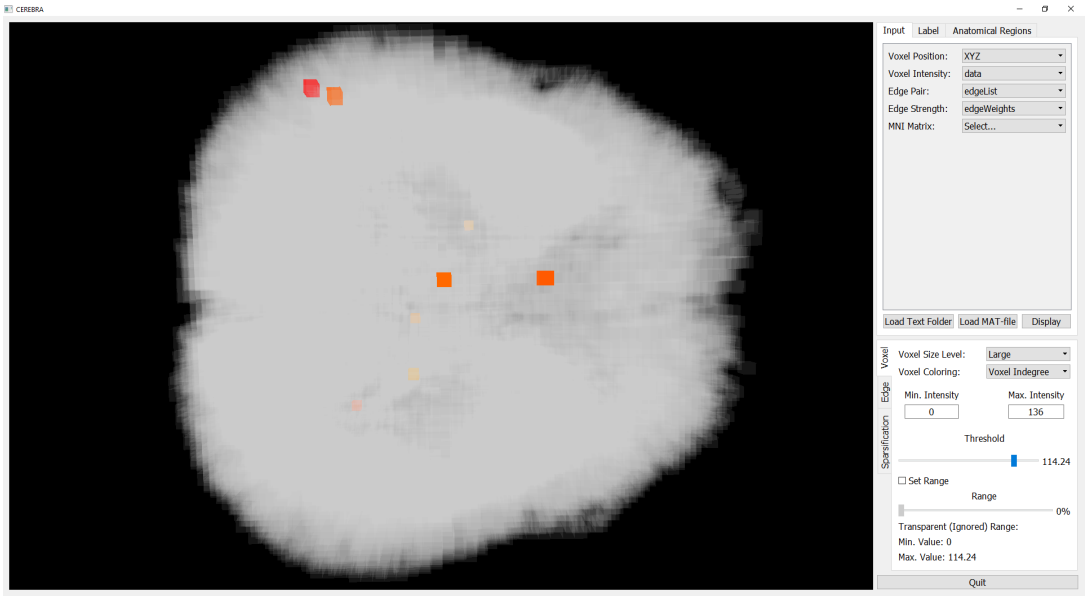


Figure 4.10: Thresholded version of the Figure 4.9. We allowed the voxels with in-degree values above 133.

filling the texture-map with $v(t, s_i)$ values for $i = 1, \ldots, N$, the algorithm puts $id(t)_i$ or $wid(t)_i$ values for $i = 1, \ldots, N$ into texture-map. Other fields, such as min-max

values, are also changed accordingly.

In-degree values for the voxels are calculated and assigned just after the user loads a data with edge information or enabling the Pearson correlation coefficient calculator of CEREBRA. An example for this colorization can be seen on Figure 4.9 for in-degree option. The input source is the same that is used on Figure 4.6. The connection is obtained using Pearson correlation coefficient where each voxel is limited to have 10 outgoing edges. Figure 4.10 is the thresholded version of the same image.

4.1.1.3 Voxel Out-Degree and Weighted Out-Degree Mapping

This mapping is similar to the voxel in-degree mapping but this time only the outgoing edges from a voxel is considered. Therefore, if a voxel makes relatively more connections compared to the others (out-degree option) or the sum of the outgoing connection weight values is higher than the overall (weighted out-degree option), then the voxel becomes more "active" than the others in this mapping. As we did in the in-degree mapping, we have employed the same coloring method with same modifications and instead of stacking voxel intensity values, $v(t, s_i)$ for $i = 1, \ldots, N$, to texture-map we have filled it with out-degree or weighted out-degree values according to selection. The min-max values are also updated with min-max (weighted) outgoing connection values. The calculation of out-degree, $od(t)_i$, and weighted out-degree, $wod(t)_i$, values for voxel i are given in the following equations.

$$od_i(t) = \Sigma_{k=1}^{N} 1_{A(t,\, s_i,\, s_k) \neq 0}, \tag{4.5}$$

where, k indicates the other voxels in the graph and $A(t,\, s_i,\, s_k)$ is the edge weight time-series of the edge emerges from voxel i and goes through to voxel k. In this equation, we add 1 to od_i if the weight between voxel i and voxel k does not equal to zero. In weighted out-degree option, we directly sum all weight values of the edges

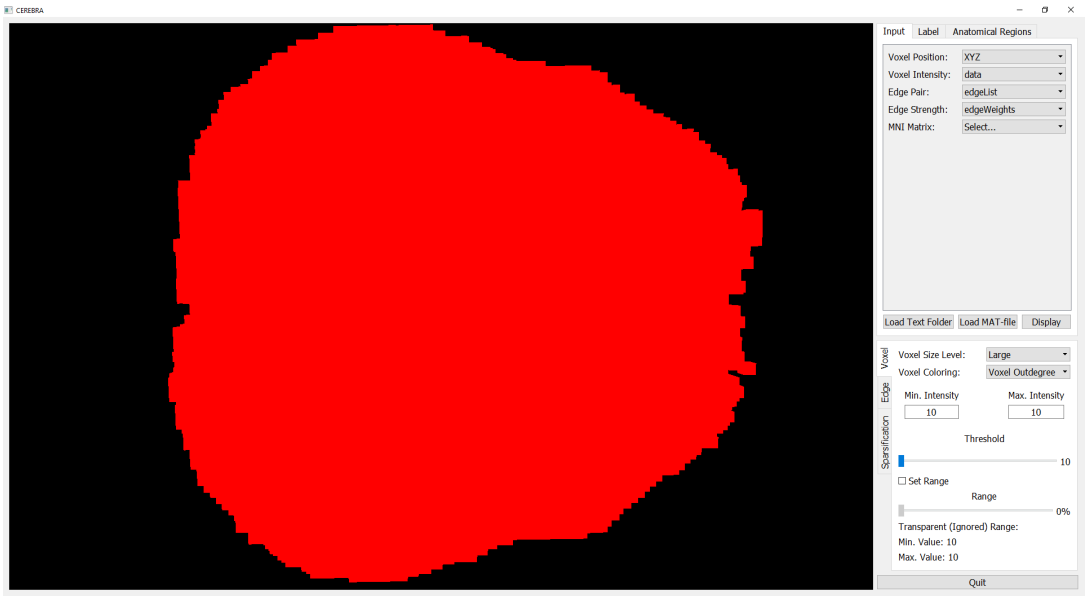


Figure 4.11: Voxel out-degree visualization on the graph illustrated in Figure 4.6. Since each edge is limited to have 10 out-going connections when Pearson correlation is calculated, all voxels have the same out-degree value. Therefore, each of them colored with red.

emerges from voxel i as follows:

$$wod_i(t) = \Sigma_{k=1}^{N} A(t,\, s_i,\, s_k). \tag{4.6}$$

Out-degree values are calculated with in-degree values and assigned just after the user loads a data with edge information or enabling the Pearson correlation coefficient calculator of CEREBRA. Out-degree visualization of the graph illustrated in Figure 4.6 is given in Figure 4.11. As a reminder, the connections are calculated using Pearson correlation coefficients and each voxel is limited to have 10 outgoing connections. Therefore, each voxel has the same out-degree value, which is 10. However, weighted out-degree value may vary from voxel to voxel as they have different relations with each other. Weighted out-degree option of the same graph is visualized in Figure 4.12.

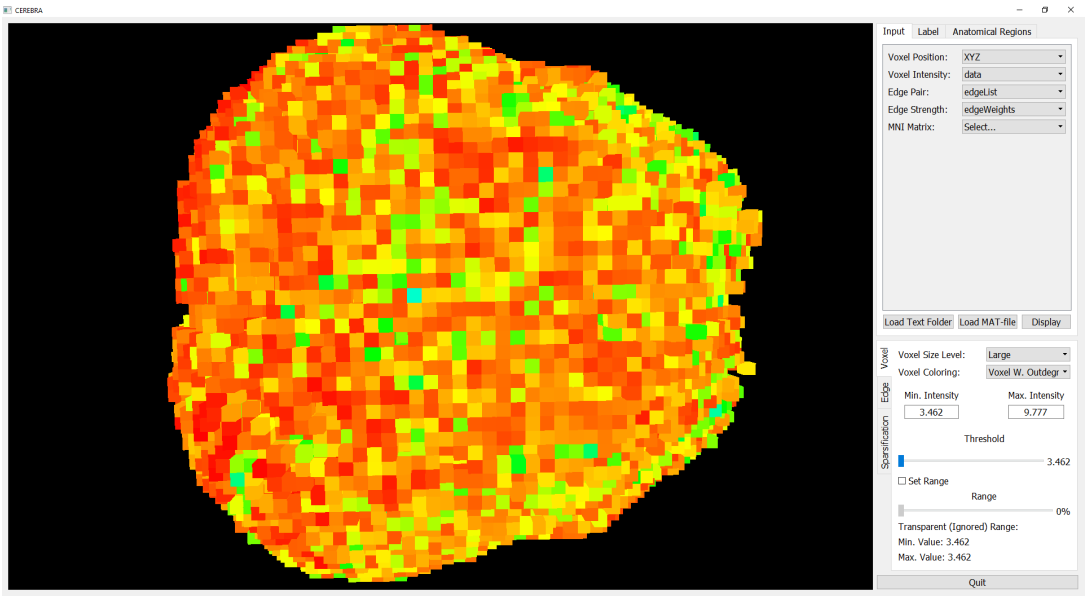


Figure 4.12: An example view from weighted out-degree mapping. Although each voxel has the same number of outgoing connections, their weights may vary according to the relations between voxels. Consequently, in this view, we can observe some green and orange voxels along with the red ones.

4.1.1.4 Voxel Total Degree and Weighted Total Degree Mapping

This mapping is the sum of in-degree and out-degree values for non-weighted option and sum of weighted in-degree and weighted out-degree values for the weighted option. When this option is enabled, the texture map is filled with the total degree values (or weighted total degree values, according to selection) and other fields in OpenGLSL function is arranged accordingly. If a voxel does not have any connection with other voxels, then it becomes colorless in non-weighted case. If the connection weights are canceled out when summed, then it becomes colorless in weighted case.

Total degree value, $td(t)_i$, and weighted total degree value, $wtd(t)_i$, of voxel i are defined as follows:

$$td_i(t) = \Sigma_{k=1}^{N} 1_{A(t,s_k,s_i)\neq 0} + 1_{A(t,s_i,s_k)\neq 0}, \tag{4.7}$$

where, the equation is the simply addition of Eq. 4.3 and Eq. 4.5. In a similar way, we can define the weighted total degree of voxel i as follows:

$$wtd_i(t) = \Sigma_{k=1}^{N} A(t,\, s_k,\, s_i) + A(t,\, s_i,\, s_k), \tag{4.8}$$

where this definition is the addition of Eq. 4.4 and Eq. 4.6.

Current method that calculates the in-degree, out-degree, total degree and their weighted versions is embedded into input reader file system and works sequentially. The algorithm is given Algorithm 4.2. Please note that, initially all voxels' od, id, wid and wod values are zero.

Algorithm 4.2 Voxel Degree Calculation

1: **for** each $voxel$ in $brain$ **do**
2: **for** each $neighbor$ in $outGoingConnections_{voxel}$ **do**
3: **if** $weight_{voxel\_to\_neighbor} \neq 0$ **then**
4: $od_{voxel} := od_{voxel} + 1$
5: $id_{neighbor} := id_{neighbor} + 1$
6: **end if**
7: $wod_{voxel} = wod_{voxel} + weight_{voxel\_to\_neighbor}$
8: $wid_{neighbor} = wid_{neighbor} + weight_{voxel\_to\_neighbor}$
9: **end for**
10: **end for**

4.1.2 Edge Coloring

Edge information is sent to the OpenGLSL function just after the voxel information is printed on the screen. Therefore, voxels and edges are printed on the display sequentially. Each edge is treated like a voxel when filling the key fields of the OpenGLSL function. Consequently, the voxel intensity texture map is stacked by the edge weight records in this case.

As in the voxel intensities, we have employed the Hot-to-Cold color scheme in or-

der to color the edges. Therefore, if the connection between two voxels is stronger compared to the remaining connections, that edge gets a reddish color. If the correlation between a voxel pair is much below the average, then that edge gets more blueish color. The mid valued connections are colored green and zero weighted edges become colorless.

Although the sequential printing method decreases the performance of the system in overall, there are no latency issues that can be understandable by the human eye.

4.1.3 Regional Coloring

In this subsection, we explain how do we handle coloring when either anatomical or functional region display options are enabled and what we offer to the users in these display options.

4.1.3.1 Anatomical Region Mapping

This option colors the voxels according to their anatomical region information. In order to mathematically relate voxels with the anatomical regions, we can define a map as follows:

$$M : s_i \rightarrow A_i, \tag{4.9}$$

where $s_i = (x_i, y_i, z_i)$ is the position of voxel i and A is the six dimensional anatomical label space. Therefore, anatomical labels are assigned according to voxels' coordinates. This mapping is one-to-one, therefore, each voxel gets an anatomical label, even if it is 0 ("Unknown").

All color codes are statically defined for six anatomical atlases and their regions as described in Section 3.3.3. These color codes are defined in "anatomicalinformation.h" header under the main project and carries all the region names mapped with their color codes. The colors are generated by taking equidistant R_a samples from the

color spectrum in Figure 4.14, for each anatomical atlases a with R_a regions. Then, the colors are randomly distributed to all regions in the atlas. In order not to assign similar colors to neighboring regions, the colors are fine tuned by hand.

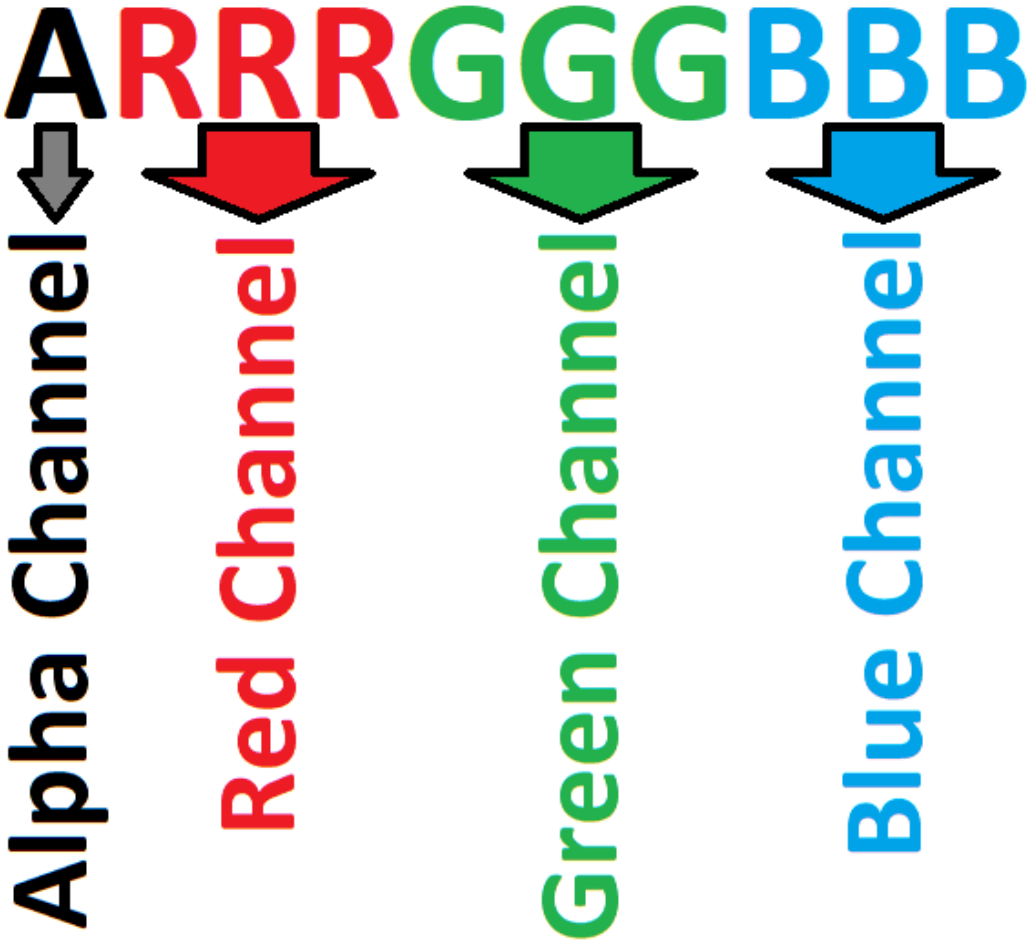


Figure 4.13: The pseudo-intensity that is sent to OpenGLSL. From left to right the concatenated number includes the alpha channel [0, 1], red channel [0, 255], green channel [0, 255] and blue channel [0, 255].

Figure 4.14: The color spectrum used in regional color assignment. Equidistant samples are taken from the spectrum, according to the number of regions in the atlas, and distributed to regions randomly.

However, as described previously in this section, the shader is designed to convert intensities to color codes, not take color codes directly and process them. Therefore, we have created a pseudo-intensity from anatomical color codes of the voxels in order not to change the structure of the shader function. The intensity value is created by concatenating the alpha and RGB values defined for each region. This number is illustrated in Figure 4.13 and the algorithm that decodes this number in shader function is showed in Algorithm 4.3. Since the pseudo-intensity number always starts with either zero or one, it could not exceed the numerical limits of C++.

Algorithm 4.3 Pseudo-Intensity Decoding and Anatomical Coloring Steps in OpenGLSL

1: $intensity = textMap[currentVoxel]$
2: $red, green, blue = 0$
3: $alpha := intensity/1000000000$
4: **if** $alpha \geq 0$ **then**
5: $\quad intensity = intensity - 1000000000$
6: $\quad red := intensity/1000000$
7: $\quad green := (intensity - (redChannel \times 1000000))/1000$
8: $\quad blue := intensity - (intensity/1000) \times 1000$
9: $\quad$ Return $vec4(red/255, green/255, blue/255, alpha)$
10: **else**
11: $\quad$ Return $vec4(-3, 0, 0, 0)$
12: **end if**

In order to enable users to track which color belongs to which region, we have displayed the region's name on the right panel with the same color that the related area is painted on the display panel. In the right panel, only the selected regions' names are displayed as other regions are displayed as ghosts on the graph at that time. An example is illustrated in Figure 4.15.

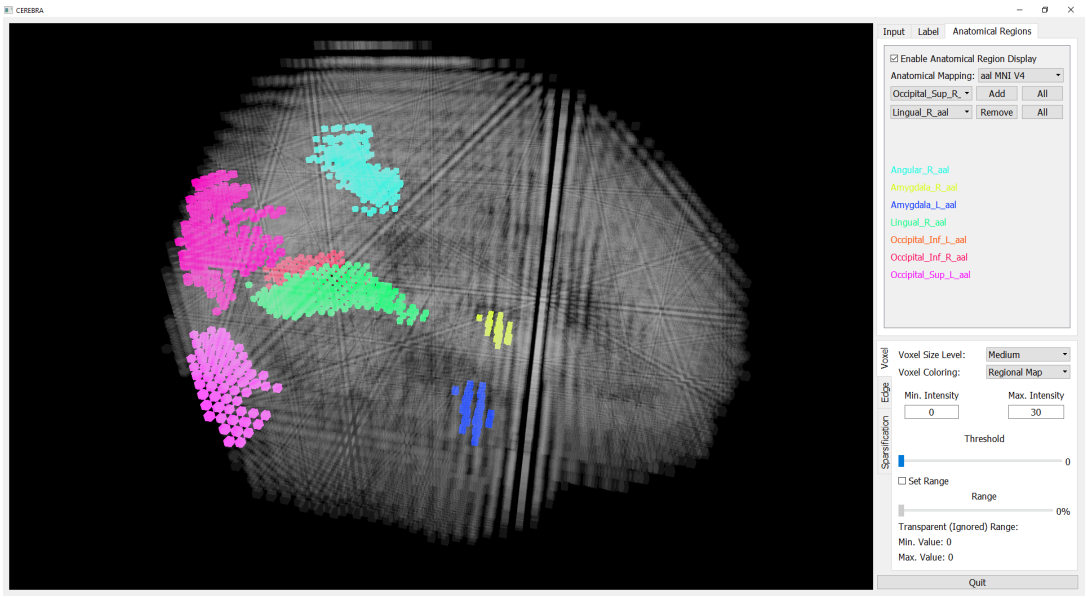


Figure 4.15: An example view from the CEREBRA where the anatomical display is enabled. In "Regional Map" option each region is displayed with a unique color and their names are indicated on the right panel with the same color.

4.1.3.2 Functional Region Mapping

Functional region mapping definition is very similar to the anatomical region coloring. We can relate voxel coordinates with functional mappings as follows:

$$M : s_i \rightarrow F, \tag{4.10}$$

where $s_i = (x_i, y_i, z_i)$ is the coordinates of voxel i and F is the multi-dimensional external label space. Since the input format of external indexing forces users to load a label for each voxel, this mapping is one-to-one.

However, since the tool could not guess the number of clusters or classes defined for an experiment, it could not assign predefined colors for regions. Therefore, it generates and assigns the color randomly whenever a region is selected by the user.

The random color assignment is done via using a map between cluster/class i.d.'s and color codes. When a label is picked up by the user, the algorithm assigns a random

color to that label and puts that matching to the map. Therefore, when a region is deactivated and activated it is matched with a new random color. The activation of a label is controlled by using another map between labels and Boolean variables. If a label is activated, the value indexed at that label becomes *true* and vice versa. The algorithm for color assignment to a label is given in Algorithm 4.4.

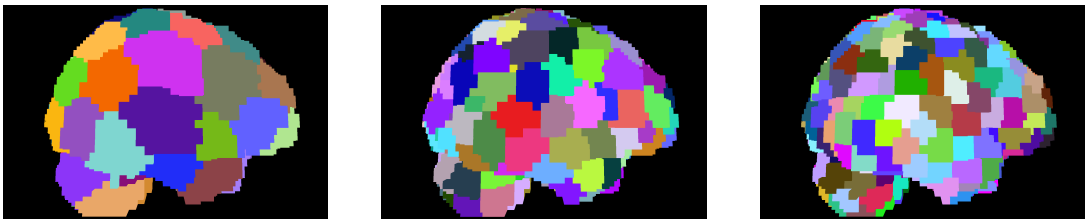
Algorithm 4.4 Assigning a Color to a Label

```
1:  procedure LABELENABLED(label)
2:      labelActivations[label] = true
3:      colorsOfLabels[label].red := random() %256
4:      colorsOfLabels[label].green := random() %256
5:      colorsOfLabels[label].blue := random() %256
6:      updateTextureMap()
7:  end procedure
8:  procedure LABELDISABLED(label)
9:      labelActivations[label] := false
10:     updateTextureMap()
11: end procedure
```

These color codes are sent to the shader as we did on displaying anatomical regions. Pseudo-intensities are prepared by the same method and they are decoded in shader function. In addition, each label is written on the right panel with the same color as they are displayed on a 3D graph. The only difference between anatomical color mapping and functional color mapping is the static and dynamic color assignment. An example is given in Figure 4.16, which shows the result of a clustering algorithm for cognitive state classification proposed by Mogultay et al. [43].

4.1.3.3 Voxel Level Intensity on Regional Display

We have given the details of the voxel intensity coloring in Section 4.1.1.1. However, what will happen when the user displays voxel intensities when either anatomical or functional display options are enabled? Under this heading, we answer this question.



(a) 100 Clusters (b) 500 Clusters (c) 1000 Clusters

Figure 4.16: Displayed result of a K-Means Clustering algorithm using functional color mapping feature. Since the colors are assigned randomly, each time the user redisplays the results, he/she will get different colors on the same labels. Images are also used in Mogultay et al.'s work [43].

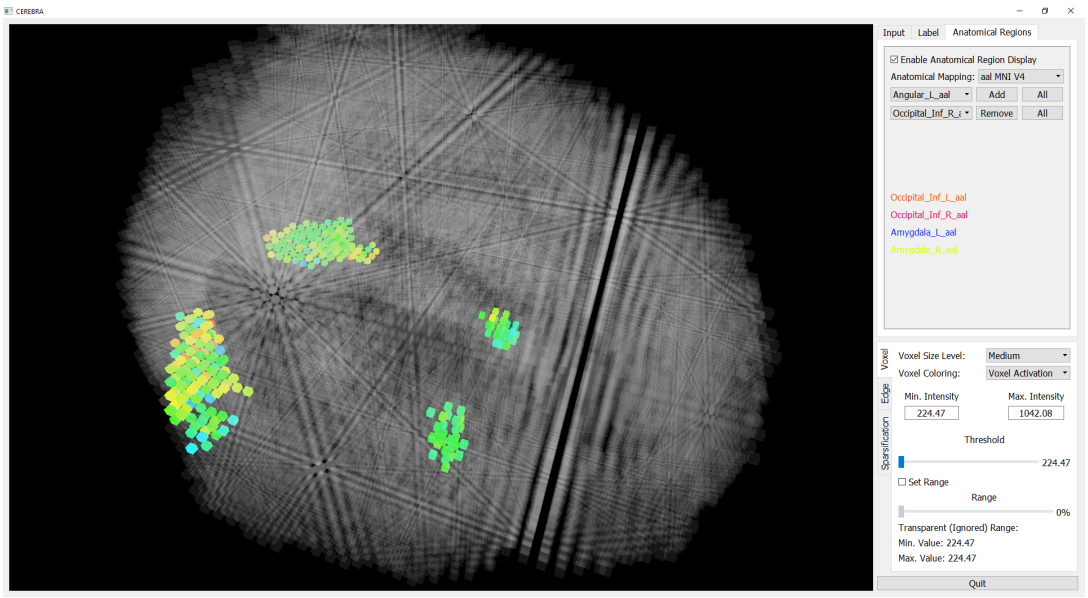


Figure 4.17: Voxel intensity visualization when region display is enabled. Four regions, namely, Occipital_Inf_R, Occipital_Inf_L, Amygdala_L, and Amygdala_R are selected. Voxel intensities of these regions are displayed, whereas rest of the regions are displayed as ghosts.

In some experiments, only a bunch of voxels belonging to a specific region(s) are important and should be tracked [17, 33, 51]. These groups may contain voxels whose intensities could be ranging from lowest values to highest ones. In this scenario, it is not possible to eliminate unwanted groups by using simple thresholders described in Section 4.5. Therefore, we have enabled users to combine anatomical displaying

with voxel intensities in CEREBRA. In other words, when regional display option is active at the same the voxel intensities are being observed, the displayed graph is updated such that only the selected regions' voxel intensity records are left on the display. Other regions immediately become colorless in order to make observation easier.

This operation is done by a simple intensity trick on texture-map. When a user enables one of the regional display options, we turn each voxels' intensity value to zero on texture-map, since initially none of the regions are selected. As the user adds regions to the display, the algorithm finds and colors the voxels belonging to the selected regions. The algorithm uses a label intensity map to accomplish this task in which each region is matched with a Boolean value. Selected regions' Boolean value turns "true" and belonging voxels' intensity records are sent to texture map without garbling their values. Unselected regions' voxels' intensity values are sent to texture map as zero to make them colorless. Each time the user activates or deactivates a region, the label intensity map and, consequently, the texture-map is updated. The logic behind this trick is given in Eq. 4.11, where $r(t)_i$ is the value pushed to texture-map for voxel i and A_i is the anatomical label of the voxel.

$$r(t)_i = \begin{cases} v(t, s_i) & A_i \in Selected\,Labels \\ 0 & otherwise. \end{cases} \tag{4.11}$$

An example view from this option is illustrated in Figure 4.17, in which, only four regions' voxel intensity records are displayed as rest of the regions are displayed as ghosts.

4.1.3.4 Average Intensity of Brain Regions

As voxel intensities within a region can be tracked, users may need to observe relations between the regions in terms of intensities. For this purpose, we have embedded an option that enables users to visualize the average intensity of regions.

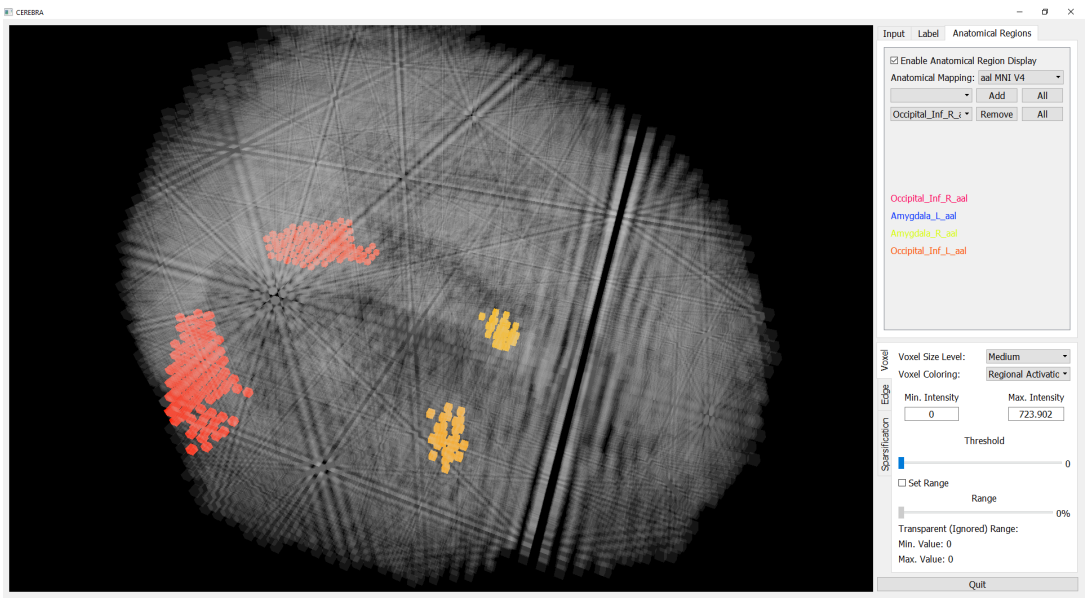


Figure 4.18: The average intensity of brain regions visualization on the same graph illustrated in Figure 4.17. The same four regions are selected and colored according to the average voxel intensities within these regions. In this view, it can be seen that occipital region is more active than amygdala.

For anatomical regions, after the labels are assigned, the average intensity value for each region is automatically calculated. This operation can only be done if the voxel intensity information is provided by the user. Otherwise, each region gets 0.5 as the average intensity value, since all voxels will have that value as default. The anatomical regions' average intensity values are held as a fixed size 3D vector in *Packet* class. The first dimension divides anatomical information into 6 atlases which are mentioned in Section 3.3.3. The second dimension stands for the regions, which lie under these anatomical atlases (e.g. Inter Hemispheric, Left Brainstem, Left Cerebellum, Left Cerebrum, Right Brainstem, Right Cerebellum, Right Cerebrum are the regions of the Hemisphere atlas.). The last dimension holds the time information, as the tool is able to animate the whole experiment. Anatomical average intensity calculation steps are introduced in Algorithm 4.5.

Since functional regions could not be preloaded to the system, we have followed a different way to calculate functional average intensity. We have another 3D vector variable in *Packet* class in order to hold the average intensity information. However,

Algorithm 4.5 Calculation of Average Intensity of Anatomical Regions

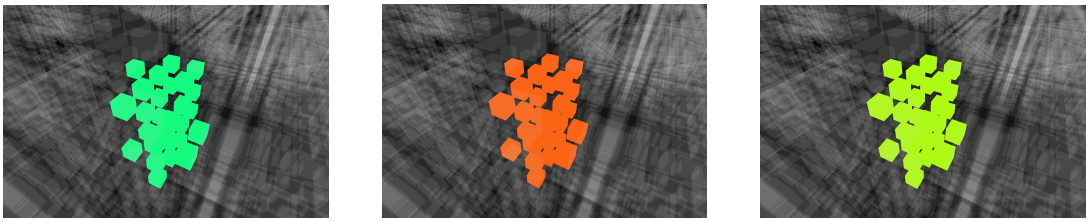
```
1: for each map in packet.anatomicalAvgIntensity do
2:     for each region in map do
3:         resize(region, Time)
4:     end for
5: end for
6: for each map in packet.anatomicalAvgIntensity do
7:     for each v in packet.brain do
8:         ar := v.anatomicalRegion
9:         while t ≤ Time do
10:            map[ar][t] := map[ar][t] + v.intensity
11:            map[ar][t] := map[ar][t]/map[ar].size
12:        end while
13:    end for
14: end for
15: for each map in packet.anatomicalAvgIntensity do
16:    anatomicalMinMaxIntensity[map] := find(min, map)
17:    anatomicalMinMaxIntensity[map] := find(max, map)
18: end for
```

this time we did not pre-allocate the memory. As assigning labels to the voxels, we store the number of labels and number of voxels under each label. Then, we resize the variable comes from *Packet* class according to the number of labels. After memory allocation is done, we added each intensity value for each time instance for each region and divide the resulting number by the number of voxels within that region. The result is the average intensity record of that label. Methods for assigning labels to the calculation of functional average intensities are given in Algorithm 4.6.

An example to this view is given in Figure 4.18, which is the same graph illustrated in Figure 4.17. The only difference is the color mapping. Selected regions are written on the right panel of CEREBRA, which are colored according to regional color mapping. It can be observed that, in this graph, occipital lobe is more active than the amygdala,

Algorithm 4.6 Functional Average Intensity Calculation

1: **procedure** ASSIGNLABELS($voxelLabelList$)
2: $labelSize :=$ Empty integer-to-integer map vector
3: resize($labelSize, voxelLabelList.size$)
4: **for** each map in $voxelLabelList$ **do**
5: **for** each $voxel$ in map **do**
6: $labelActivations[map[voxel]] := false$
7: push_back($voxel.functionalLabel, map[voxel]$)
8: $labelSize[map][map[voxel]] := labelSize[map][map[voxel]] + 1$
9: **end for**
10: **end for**
11: fingLabelAvgIntensity($labelSize$)
12: **end procedure**
13: **procedure** FINDLABELAVGINTENSITY($labelSize$)
14: resize $packet.functionalAvgIntensity$, using number of maps and region information.
15: **for** each map in $packet.functionalAvgIntensity$ **do**
16: **for** each $voxel$ in $packet.brain$ **do**
17: fr := voxel.funtionalRegion
18: **while** $t < Time$ **do**
19: $map[fr][t] := map[fr][t] + voxel.intensity$
20: $map[fr][t] := map[fr][t]/map[fr].size$
21: **end while**
22: **end for**
23: **end for**
24: **for** each map in $packet.functionalAvgIntensity$ **do**
25: $functionalMinMaxIntensity[map] :=$ find(min, map)
26: $functionalMinMaxIntensity[map] :=$ find(max, map)
27: **end for**
28: **end procedure**



(a) Regional Within Degree (b) Regional Inter Degree (c) Regional Total Degree

Figure 4.19: An example of regional degree display on voxels. The displayed region is the Amygdala_L from AAL atlas.

since it is closer to red than amygdala.

4.1.3.5 Within Degree of Brain Regions

Brain parcellation algorithms based on clustering (such as N-Cut [67], K-Means [41] etc.) could be evaluated by looking at cluster mixture density functions. The density could be checked by the number of connections within a brain region. If all voxels within that region make many mutual interactions, then the algorithm could stop further segmenting that cluster. In some cases, this decision could be rechecked and changed by the researcher. However, viewing all connections in a small and dense region may hide some important information behind the connection mess. Therefore, we have put an option to color each region according to their total number of self-connections, in other words, within degrees.

Similar to the calculation of functional average intensities, functional within degree is also calculated after label information is loaded and if connection information is available. Then, for each voxel, the algorithm looks for its neighbors. If the voxel is connected by another voxel which has the same label, then we count that connection as a within connection. Therefore, that label's within degree value is incremented by one. These degree values are collected in a 3D vector called, *functionalWithinDegree* in the *Packet* class. Whenever a label is selected to be displayed, we fetch the degree value from that vector and update texture map accordingly to color that region.

Although this algorithm is designed for functional regions, it could be used with the anatomical mapping feature. The anatomical within degrees are calculated just after the data is loaded to the CEREBRA, if connection information is available. Anatomical within degree values are stored in a variable in the *Packet* class, defined as a fixed-sized 3D vector, named *anatomicalWithinDegree*. If two connected voxels belong to the same anatomical region, then the algorithm increments that anatomical region's within degree by one.

4.1.3.6 Inter Degree of Brain Regions

Brain parcellation algorithms, based on clustering, can also be evaluated by checking the balance between connections within regions and between regions. Therefore, users may also want to track regions' outer connection degrees. In this view, all regions are colored by their outer connection degrees.

This functionality can also be used to determine hub regions. These regions are the ones that ensure the communication between other regions (e.g. important people in social media). In this view, the red regions indicate the hub regions which may play a key role in the monitored experiment.

Regional inter degree is calculated with regional within degree. If two connected voxels do not belong to the same region, then the algorithm increments the related field of the regional inter degree variable by one. This variable lies in the *Packet* class as a fixed sized 3D vector form for anatomical regions and un-allocated 3D vector form for functional regions.

4.1.3.7 Total Degree of Brain Regions

We have embedded it in the tool if users want to observe regions/clusters that are both denser and out-connected than the others.

Total degree calculation is done by adding regional inter degree and within degree values. The algorithm that calculates these values are given in Algorithm 4.7. Since

the only difference between anatomical and functional versions of these calculations is determining the number of voxels in a functional region and allocate the memory accordingly, which is shown in Algorithm 4.6, we combined functional and anatomical degree calculation in one algorithm in this subsection. In addition, an example for these three regional degree coloring option is illustrated in Figure 4.19.

4.1.3.8 Within Edge Colorization and Display of Brain Regions

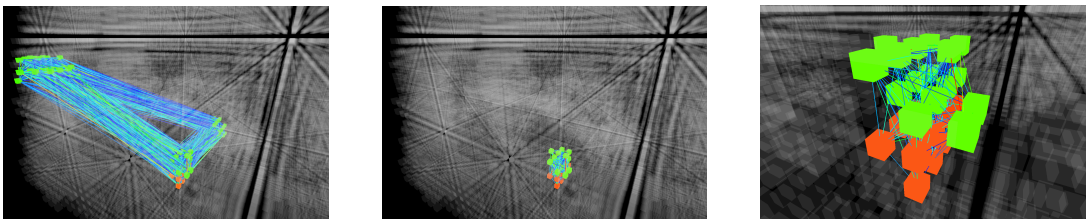
Although observing degree values from the voxel colors is sufficient for many cases, sometimes scientists need to directly observe the relations among the voxels in a specific region as a graph in order to make some conclusions. For example, change of the patterns in connections within a region may help to determine whether the subject is healthy or not [14, 55, 69]. Another case may be discrimination of two similar tasks that take places in the same brain regions [37, 71]. In such cases, it would be difficult to observe edges that belong to only a specific area among all. Therefore, we have enabled users to select regions to display only those regions' self-connections.

In order to accomplish this task, we have employed the methods that are mentioned in Section 4.1.2 and Section 4.1.3.3. In those methods, the algorithm first assigns their region labels to voxels. Then, using a label-Boolean map, it determines which labels' edges are displayed. Initially, each label is assigned to *false*, which indicates initially no edges will appear on display. As the user selects regions, the corresponding Boolean variable in the map turns to *true* which enables self-connections of the selected regions on display. This activation/deactivation operation is done via changing the weights of the related edges. The algorithm checks each edge whether their two sides belong to the same region and, if so, passes it's weight as it is. If they do not belong to the same region, the algorithm automatically eliminates that edge from the display by pushing the weight value to texture map as zero. The difference between standard edge display and regional within display is showed in Figure 4.20.

Algorithm 4.7 Regional Degree Calculation

```
1: #rwd: regionalWithinDegree, rtd: regionalTotalDegree, rid: regionalInterDegree
2: procedure FINDREGIONALDEGREE(regionSize)
3:     for each map in packet.rwd do
4:         for each v in packet.brain do
5:             for each c in v.outGoingConnection do
6:                 n := c.neighbor
7:                 l1 := v.functionalLabel[map]
8:                 l2 := n.functionalLabel[map]
9:                 if l1 = l2 then
10:                    for each t in Time do
11:                        if c.weight ≠ 0 then
12:                            packet.rwd[map][l1] + +
13:                            packet.rwd[map][l1] : / = regionSize[map][l1]
14:                            packet.rtd[map][l1] + +
15:                            packet.rtd[map][l1] : / = regionSize[map][label1]
16:                        end if
17:                    end for
18:                else
19:                    for each t in Time do
20:                        if c.weight ≠ 0 then
21:                            packet.rid[map][l1] + +
22:                            packet.rid[map][l1] : / = regionSize[map][l1]
23:                            packet.rtd[map][l1] + +
24:                            packet.rtd[map][l1] : / = regionSize[map][l1]
25:                        end if
26:                    end for
27:                end if
28:            end for
29:        end for
30:    end for
```

```
    for each map in packet.rwd do
        packet.regionalWithinDegreeMinValue[map] := find(min, packet.rwd)
        packet.regionalWithinDegreeMinMaxValue[map + 1] := find(max, packet.rwd)
        packet.regionalInterDegreeMinMaxValue[map] := find(min, packet.rid)
        packet.regionalInterDegreeMinMaxValue[map + 1] := find(max, packet.rid)
        packet.regionalTotalDegreeMinMaxValue[map] := find(min, packet.rtd)
        packet.regionalTotalDegreeMinMaxValue[map + 1] := find(max, packet.rtd)
    end for
end procedure
```



(a) All Edges (b) Regional Within (c) Regional Within Zoomed

Figure 4.20: The effect of within region edge display. Figure 4.20a shows all the connections placed within and between the regions. In 4.20b is the same graph with only one region and its within connections. The user may zoom-in to observe connections better as shown in 4.20c.

4.1.3.9 Inter Edge Colorization and Display of Brain Regions

As in the examples given in the section above, the scientists may also need to observe connections and their weights to make some conclusions about the cognitive experiment performed during an fMRI recording. For example, strongly connected regions

may indicate that those regions work together to accomplish the given cognitive task in that experiment [70, 8]. In addition, it is known that by tracking the resting state fMRI experiments and relation between some specific regions, scientists could diagnose diseases such as schizophrenia it alters the connection structures between some specific regions [66, 75]. Therefore, with regional within edge degree colorization, we also add this feature to enhance CEREBRA's capabilities.

This algorithm works as a complement to the within edge colorization of brain regions as it colorizes the edges which are eliminated by that algorithm and eliminates the selected ones by it. In other words, after labels are assigned, it looks for the connections of voxels in selected regions. If the other side of the connection lies in the same region, then the algorithm sets that connection's weight value to zero in the texture map. If two sides of the edge belong to different regions but the edge emerges from a non-selected region, then it is again eliminated from the display. However, if two sides belong to different regions and the edge emerges from the selected region, then the weight of that edge will pass from the algorithm.

The formal version of the algorithms for both within and inter edge colorization of brain regions are given in Algorithm 4.8.

4.1.4 Animating Voxel and Edge Colors

As mentioned in Chapter 2, an fMRI machine shoots a full brain volume in every 2 to 3 second. In other words, the voxel intensity values are updated in every 2 to 3 second. Therefore, displaying only a part (shoot) of the experiment may constrain the users from observing consecutive intensity changes occurring in the course of the experiment. In order to solve this problem, we have implemented animation functionality that is able to animate all of the coloring options mentioned in this section. The animation is done via linear interpolation of two sequential intensity/weight values and using OpenGL functions.

First, the input should include the voxel intensity or edge weight change in time, as described in Section 3.2.2 and Section 3.2.4. If both voxel intensities and edge

Algorithm 4.8 Within and Inter Edge Degree Colorization of Brain Regions

```
1: map := Displayed region map
2: if Regional Display Active then
3:     for each v in packet.brain do
4:         for each connection in v.outGoingConnection do
5:             neighbor := connection.neighbor
6:             if Regional Within Edge Colorization Available then
7:                 if v.regionLabel[map] = neighbor.regionalLabel[map] then
8:                     while t ≤ Time do
9:                         push_back(textureMap, connection.weight)
10:                    end while
11:                else
12:                    while t ≤ Time do
13:                        push_back(textureMap, 0)
14:                    end while
15:                end if
16:            else if Regional Inter Edge Colorization Available then
17:                if v.regionLabel[map] ≠ neighbor.regionalLabel[map] then
18:                    while t ≤ Time do
19:                        push_back(textureMap, connection.weight)
20:                    end while
21:                else
22:                    while t ≤ Time do
23:                        push_back(textureMap, 0)
24:                    end while
25:                end if
26:            else
27:                while t ≤ Time do
28:                    push_back(textureMap, connection.weight)
29:                end while
30:            end if
31:        end for
32:    end for
33: end if
```

Table 4.1: Interpolation level and update frequency definitions in CEREBRA and their values.

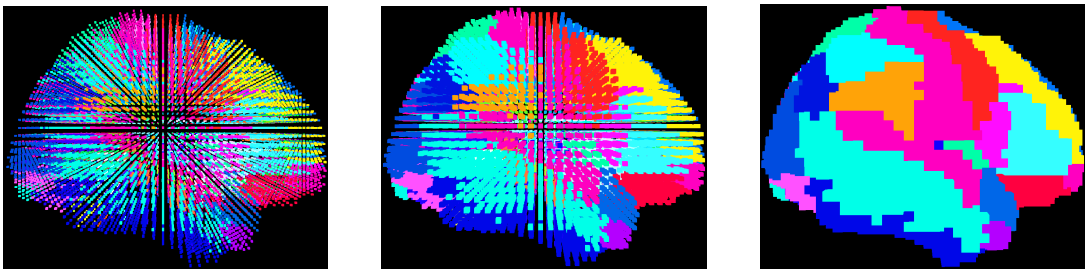
Define	**Value**
INTERPOLATION_LEVEL	40
UPDATE_FREQ_IN_MS	50

weights have time-series information, they should last for the same amount of time (e.g. both should have the duration of 5 minutes). Otherwise, CEREBRA will give an error and do not process the input file. If the input format is correct, then time-series information is loaded to *Packet* class. After the data pass through the preprocessing steps, the information is transferred to OpenGLSL function. The texture map, carries the intensity and weight information in OpenGLSL, is loaded with time-series information of these records. Since the texture map is one dimensional, we have stacked each time vector of the following voxels, one after another. Since we know the length of the experiment, at each call of *UpdateGL()* function, which updates the display, we basically skip the next intensity value and update display accordingly.

However, this sharp transition between intensity values may confuse the users and make it difficult to follow changes. Instead of displaying each intensity value on the display during 2 seconds and then switching the next value immediately, we have divided the time between these two values into small pieces. In other words, we increase the number of updates by calling *UpdateGL()* function more frequently. At each call, we approximate the next intensity value by a step, whose size is determined and defined by us. In CEREBRA, we define the number of small times slices between two consecutive intensity values to 40 and update time to $50ms$. In other words, the display is updated in every $50ms$ and we update the display 40 times to reach the next intensity value. Therefore, it takes 2 seconds to reach the next intensity value which is consistent with a typical fMRI machine's update time. These values are defined in renderer as shown in Table 4.1 with their names. In order to approximate the next intensity value step by step (make interpolation work), we need to update Algorithm 4.1 as Algorithm 4.9. Updated parts are indicated with red color.

Algorithm 4.9 Updated Coloring Algorithm in OpenGLSL for Animation

1: $CurrentIntensity := textMap[currentVoxel + textureOffSet]$
2: $NextIntensity := textMap[currentVoxel + (textureOffSet + 1)\%Time]$
3: $intensity := CurrentIntensity + interpolationOffset \times ((nextIntensity - curIntensity)/(interpolationLevel))$
4: $minValue :=$ The lowest voxel intensity value in the loaded data
5: $maxValue :=$ The highest voxel intensity value in the loaded data
6: $r := 0.0$
7: $rMin := 0.0$
8: $rMax := 1.0$
9: $minMaxDistance := 0.0$
10: $red, green, blue := 0$
11: $r =$ Normalize$(intensity, minValue, maxValue)$
12: **if** $minValue \leq 0$ **then**
13: $\quad rMax := maxValue/(maxValue - minValue)$
14: $\quad rMin := minValue/(maxValue - minValue)$
15: $\quad minMaxDistance := rMax - rMin$
16: **else**
17: $\quad minMaxDistane := 1.0$
18: **end if**
19: **if** $r < (rMin + 0.25 \times minMaxDistance)$ **then**
20: $\quad red := 0$
21: $\quad green := 4 \times (r - rMin)/minMaxDistance$
22: **else if** $r < (rMin + 0.50 \times minMaxDistance)$ **then**
23: $\quad red := 0$
24: $\quad blue := 1 + 4 \times (rMin + 0.25 \times valueDistance - r)/valueDistance$
25: **else if** $r < (rMin + 0.75 \times minMaxDistance)$ **then**
26: $\quad blue := 0$
27: $\quad red := 4 \times (r - rMin - 0.5 \times valueDistance)/valueDistance$
28: **else**
29: $\quad blue := 0$
30: $\quad green := 1 + 4 \times (rMin + 0.75 \times valueDistance - r)/valueDistance$
31: **end if**
32: Return $vec4(red, green, blue, 1.0)$



(a) Small (b) Medium (c) Large

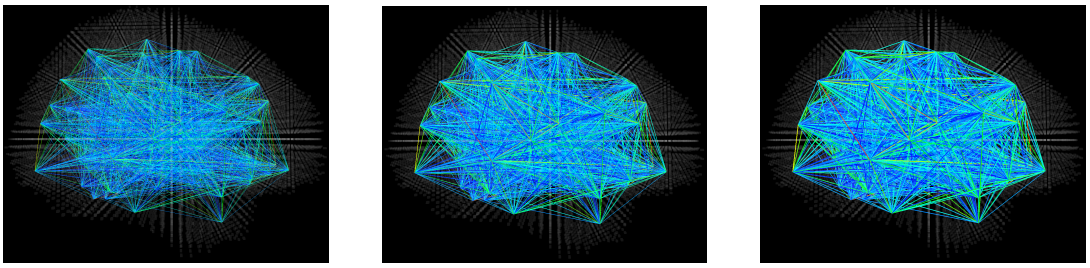
Figure 4.21: Comparison of different voxel size levels on the same graph.

4.2 Managing Voxel Size and Edge Thickness Levels

As we mentioned before, fMRI machine divides the brain into small cubes, which we call voxels. In the raw fMRI data, there are no spaces between voxels. However, this situation makes it difficult to observe intensity diversity and connections between voxels. Therefore, the tool initially reduces the dimensions of the voxels to put some space between them.

Although the initial size of the voxels is useful for many cases, it can be changed by the user into one of the three different levels, namely, small, medium and large. Small sized voxels are useful when the user works on connections rather than voxel activities. In this view, the tool shrinks each side of the voxels to 0.250 units. Medium is the default size of the system and is useful when both voxels and their connections are studied by researchers. Medium voxels have sides that are 0.500 units. Large size is the original size of the fMRI data in which there are no spaces between voxels. This size is appropriate when the user wants to observe the outer shell activities or there is no connection between voxel groups. As it is the original size of the voxels, they all have sides of 1.00 unit. The differences between the three voxel sizes are shown in Figure 4.21.

Like voxel sizes, edge thicknesses can also be modified in CEREBRA. To modify the edge thicknesses, the user should select the appropriate thickness level from the tool. There are three levels are available, namely, thin, normal and thick. When there



(a) Thin (b) Normal (c) Thick

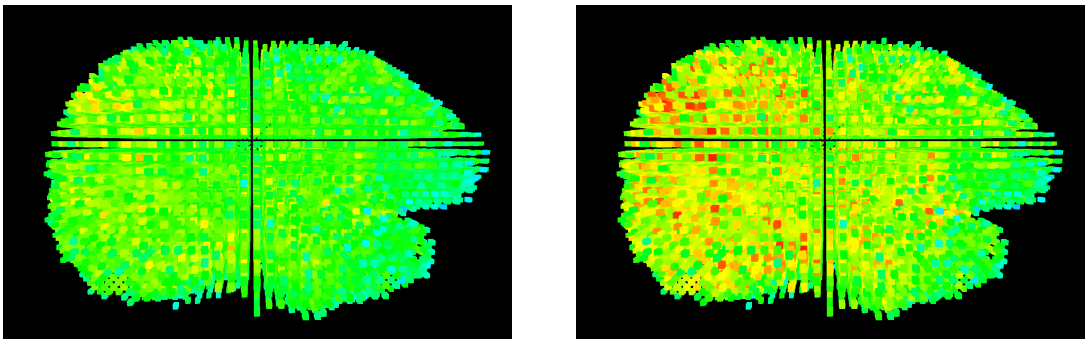
Figure 4.22: Comparison of different levels of edge thicknesses on the same graph.

are many edges on the display, setting the edge thickness level to thin may help to discriminate edges. In this view, each edge has the thickness of 1.0 unit. Normal is the default level of the thickness and useful when both voxels and edges are studied. Normal sized edges have the thickness value of 2.0. The thick level is recommended when there are not many edges on the display and the user mainly works on the connection information. Thick edges have 3.0 thickness unit. The level differences are illustrated in Figure 4.22.

4.3 Normalizing Voxel Intensity and Edge Weights In a Range

CEREBRA automatically adjusts the mapping between intensity/weight records with the color scheme. The calculation is done in OpenGLSL function as given on Algorithm 4.1 (line 9) and its updated version Algorithm 4.9 (line 11). Since the normalization is done by using directly the lower and upper boundaries of the subject, we have different scales for each individual and for each experiment even if the subject is the same. For this reason, CEREBRA allows users to set global boundaries on these intensity/weight records. The color codes rearranged by using the new boundaries and color mapping will be updated on the display. In this way, all subjects are colored using the same color scale and researchers can observe the inter-subject relations easily.

As data is loaded into the system, the algorithm finds global minimum and maximum for all possible coloring options (intensity, weight, degree etc.) and stores them in

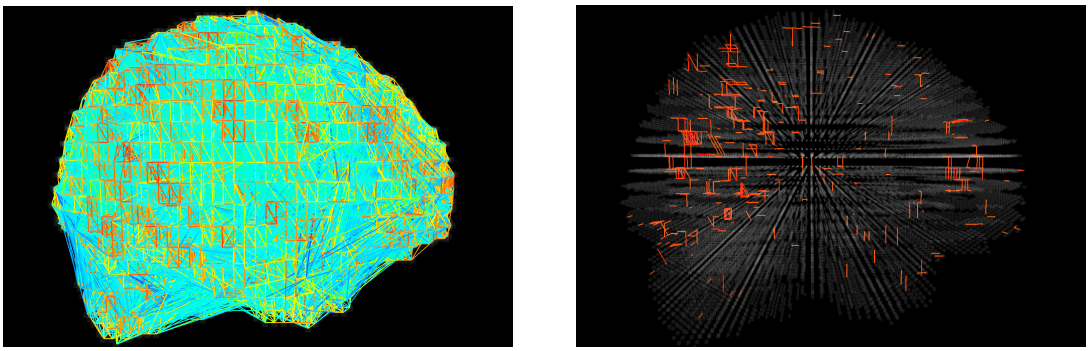


(a) Original (b) Re-Normalized

Figure 4.23: Difference between the original and re-normalized graph. In (a), some outlier voxels cause all others to be placed in mid-frequencies. Ignoring outliers on normalization step, that is updating global min and max values according to the intensity histogram and disregarding the outliers, makes discrimination of voxel intensities easier and the graph becomes more understandable as can be seen in (b).

related fields of the *Packet* class. When the user changes the coloring option, the related min-max values are sent to OpenGLSL function in order to do normalization, and eventually coloring, correctly. Using the input boxes under "Voxel" and "Edge" tabs, the user could change these min-max values and set new range for normalization operation. New values are sent to OpenGLSL function dynamically as the user changes the values.

Besides from multi-subject studies, changing the normalization range could be useful when there are outliers in intensity, weight or degree values. These values cause other high intensity/weight/degree values to be colored as if they are in the mid range. In order to cope with this problem, the min-max values could be rearranged according to the histogram of the intensities/weights/degrees. An example situation could be seen in Figure 4.23, in which all voxels are seemed to have near intensity values initially, because of the outliers. However, when the min-max values for the voxel intensity values are rearranged, we can see that there are significant distinctions between intensities.



(a) p=10, unprocessed

(b) p=10, processed

Figure 4.24: The result of the Pearson correlation coefficient calculation. The first graph is the direct result of the algorithm, in which each voxel has 10 outgoing edges. The second graph is thresholded to observe the edges with higher weights.

4.4 Pearson Correlation Coefficient Calculator

As mentioned before, the connectivities among the voxels or anatomic regions may carry more information than the voxel-wise or regional activities about the underlying cognitive process. Therefore, to enable users to benefit from this information, we have embedded a simple network estimation algorithm, based on Pearson correlation. The algorithm calculates the pairwise correlations between voxel pairs and forms the functional connectivity matrix.

However, considering the fact that the number of voxels in a typical fMRI experiment is 10^5 to 10^6. Pearson correlation brings approximately 10^{10} to 10^{12} edges to the display, which results in a very cluttered appearance (hairball problem) of the brain network. In order to solve this problem, we suggest a user interactive method, where the user limits the number of edges depending on his/her specific study. This task is achieved by setting a global threshold on the number of edges. This way, he/she selects the specific number of connections among the strongest ones. Consequently, the output graph becomes a sparse representation of the brain network where only the strongly correlated voxels' edges are displayed.

To use this feature of CEREBRA, the user should specify a node degree limit by

using the input box on "Edge" tab, next to "Display Pearson Correlation" tag. After the limit is set, the user should click the checkbox placed in the same line (left side of the tag). Then, CEREBRA calculates the correlation between all voxel pairs and display the edges whenever the calculation is done. Edges, calculated by this feature, can be removed by unchecking the same checkbox. The calculation steps are shown in Algorithm 4.10 and Figure 4.24 shows an example output of this method, where the total of ten outgoing connections for each voxel are displayed.

Algorithm 4.10 Pearson Correlation Coefficient Calculation

```
1:  procedure FINDCORRELATION(x, y)
2:      c := covariance(x, y)
3:      cor := c/(σ_x × σ_y)
4:      Return cor
5:  end procedure
6:  procedure CALCULATEPEARSON(p)
7:      for each v in packet.brain do
8:          resize(v.outGoingConnections, size(packet.brain.size() − 1))
9:          for each n in packet.brain except v do
10:             c := findCorrelation(v.intensity, n.intensity)
11:             push_back(v.outGoingConnections, n)
12:         end for
13:         sort(v.outGoingConnections.weight)
14:         if size(v.outGoingConnections) > p then
15:             resize(v.outGoingConnections, p)
16:         end if
17:     end for
18: end procedure
```

4.5 Basic Thresholding on Voxel Intensity and Edge Weight Records

As mentioned in the Section 2.4.1, when there are many edges on the display it becomes harder to observe the specific intensity groups on the display. In such situa-

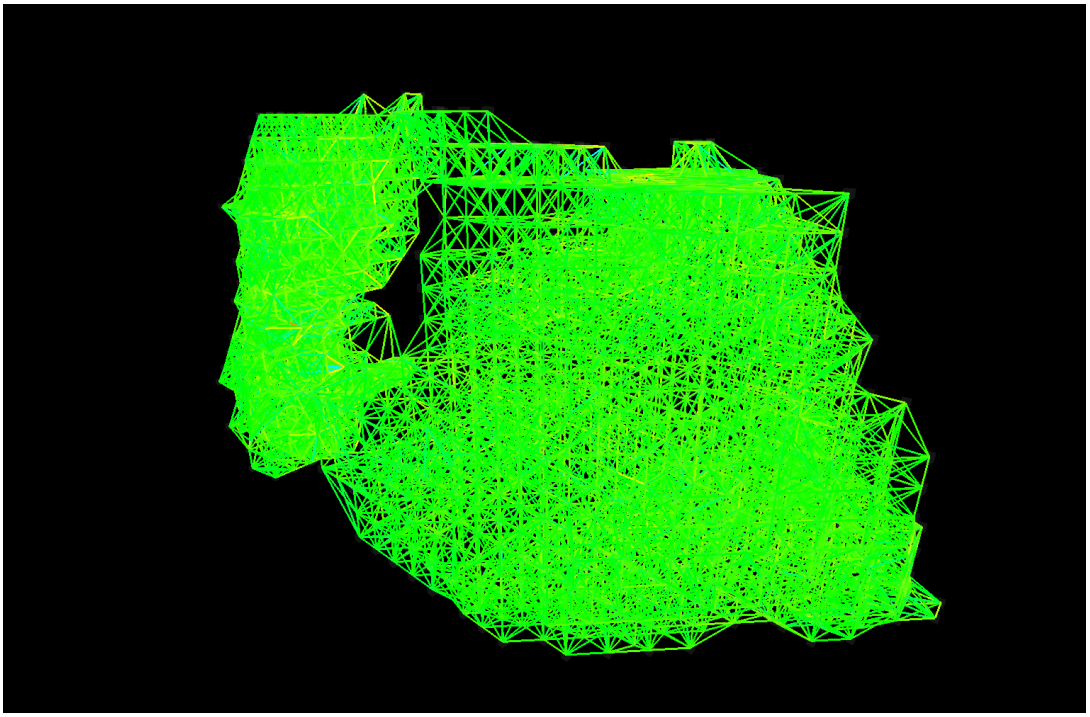


Figure 4.25: 3D view of occipital lobe. In order to observe edges easily, voxels are removed from the view. Edges are unprocessed and weight values vary between -0.6 and 0.7.

tions, the user should use basic thresholding functionality of CEREBRA. Since the background of thresholding is explained in Section 2.4.1, we will only mention how the thresholding is used on CEREBRA, and give some examples of their results on brain graphs under this heading.

In order to use this feature, the user should set a threshold value (or two values for ranged ones). This threshold value does not change across the time. Therefore, if the voxel or/and edge time-series are provided in the data and one of the thresholding methods mentioned below is applied, the voxels or edges may appear and disappear during the experiment simulation. This happens because the fluctuations in voxel intensity values may exceed the threshold level(s) on some part of the experiment and stay under the threshold during the rest. Since, voxel intensity values fluctuate during the experiment, the value of the bond between them too. Edges may "flick" as their weights change across time. The threshold values are applied by adding a

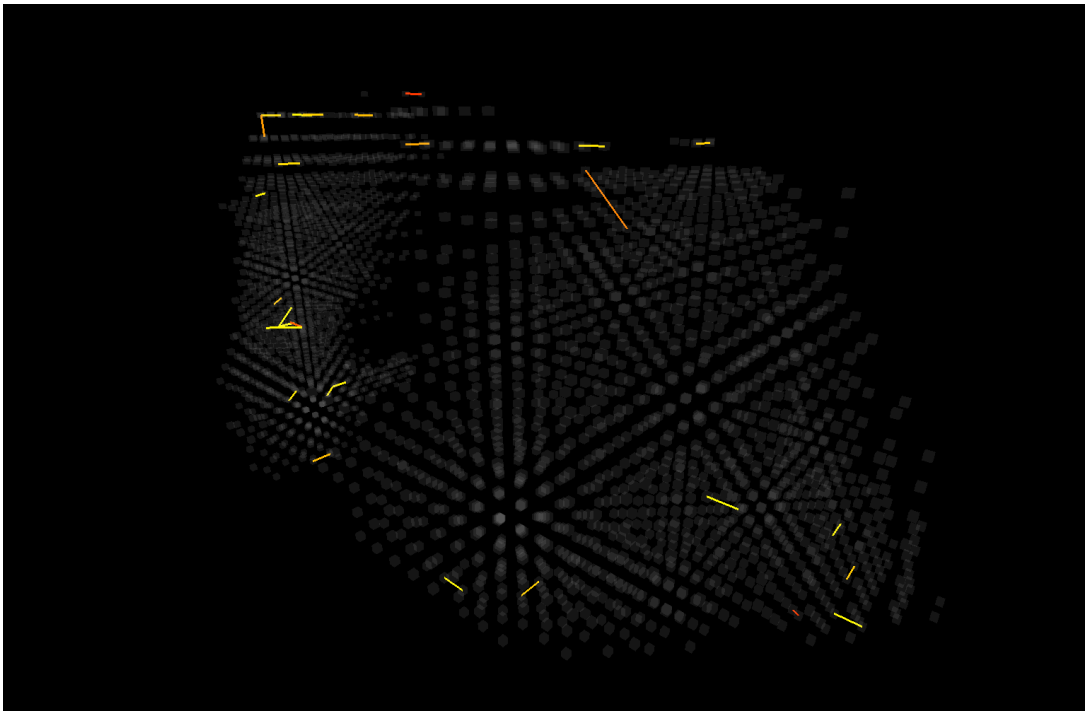


Figure 4.26: High-pass thresholding on the occipital lobe graph. The band blocks the edges with weight 0.38 and below.

simple "if" control statement to the OpenGLSL function. The section continues with the examples of each threshold type and their effects on the same graph shown in Figure 4.25.

- **High-pass Thresholding:** This method attenuates the less activated voxels and leaves the most active ones on display. To use this feature on voxel intensities, the user should use the "Threshold" slider placed within the "Voxel" tab. The number on the right side indicates the lowest displayed intensity during the experiment. Initially, this number equals to the minimum intensity value. As the user moves the slider to the right, the number converges to the highest intensity value of the voxel intensity. Voxels with lower intensity values than the threshold (the number next to slider) become ghost voxels.

  The usage of this threshold on edge weights is similar to the voxels. This time the user should use the "Threshold" slider under the "Edge" tab. As the user moves the slider to the right, the threshold value increases and the values below

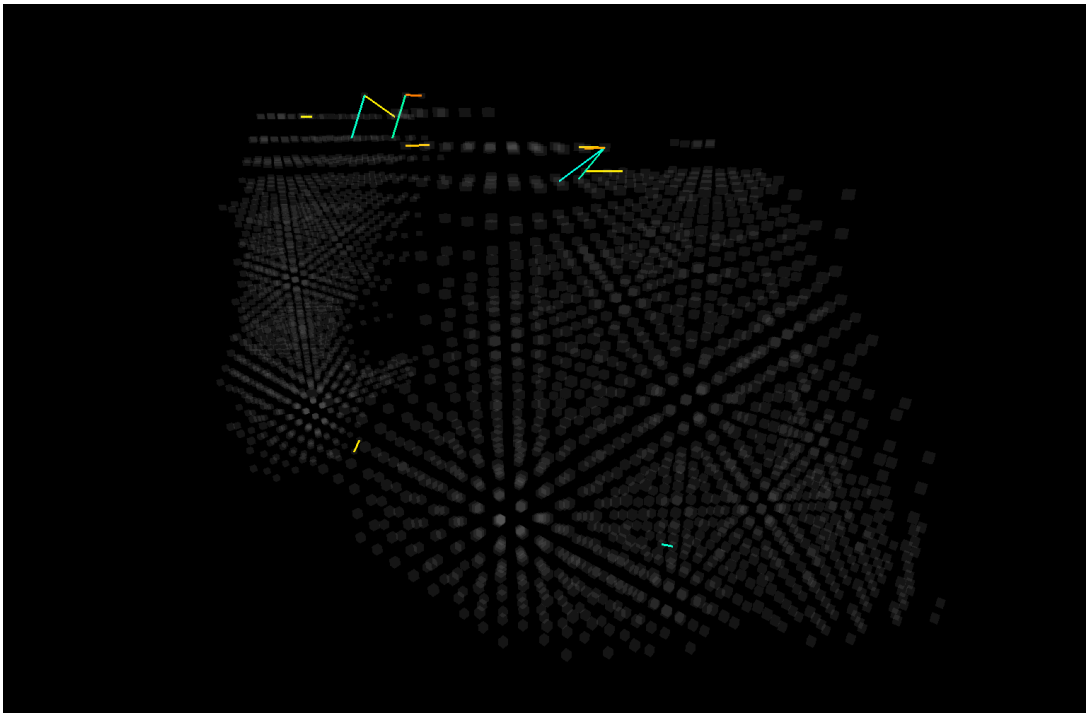


Figure 4.27: Band-stop thresholding on the occipital lobe graph. Weights within -0.15 and 0.38 are blocked. Since this is a time-series data, we may have more or less edges on display on different time instances.

that threshold are eliminated from the display, as seen on Figure 4.26.

- **Band-stop Thresholding:** This thresholding type filters a specific intensity range while allowing the outsiders to pass. To apply a band-stop thresholding on the displayed brain graph, the user should enable the *range* functionality of thresholding. This can be done by checking the "Set Range" checkbox placed under the "Voxel" and "Edge" tabs. Once this feature is enabled, two sliders (Labeled as "Threshold" and "Range") start to work coordinated.

  To eliminate the mid values, the user should drag both sliders to the middle position. Then, according to intensity distribution, the user may calibrate the sliders for the best result. If this method is applied to the edges as shown in Figure 4.27, only the red, and blue connections remain in the display (of course it is subject to filter size).

- **Low-pass Thresholding:** Similar to the Band-stop thresholding, to apply a

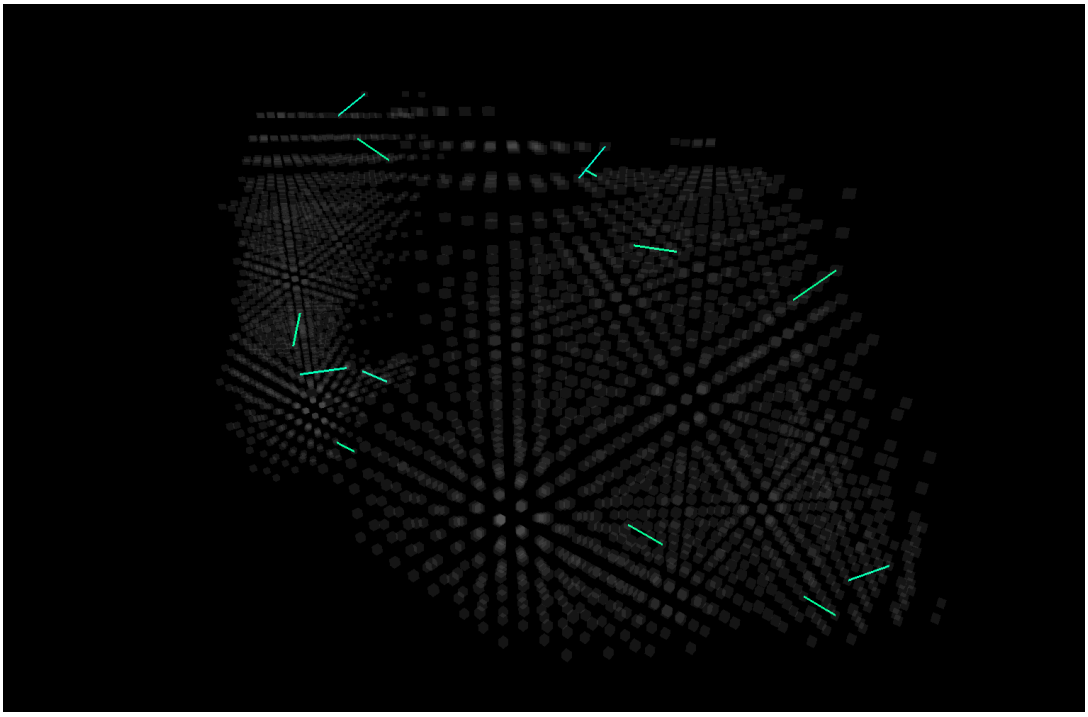


Figure 4.28: Low-pass thresholding on the occipital lobe graph. Edges with weight values above -0.13 are blocked.

low-pass threshold on the displayed brain graph, the user should enable the *range* functionality. This can be done by checking the "Set Range" checkbox placed under the "Voxel" and "Edge" tabs. Once this feature is enabled two sliders (Labeled as Threshold and Range) start to work together.

To eliminate the mid and high intensity values, the user should drag both sliders to the max position. Then, the "Range" slider begins to work as the low-pass threshold. However, the user should use it from right to left. As the user drags the slider to the left, higher values are getting allowed on the display. A simple usage of this thresholding on edges is illustrated in Figure 4.28.

4.6 Local Graph Sparsification

The edge sparsification method suggested in this study is an extended version of Satuluri et al.'s work [63]. The proposed algorithm suppresses the edges according to

a local threshold calculated for each voxel separately, rather than applying a global thresholding criterion to the whole graph. For this purpose, the similarity between voxel pairs is calculated using Jaccard coefficients.

Mathematically, Jaccard coefficients are defined between voxels i and j as follows:

$$Sim_J(i,\, j) = \frac{|Adj(i) \cap Adj(j)|}{|Adj(i) \cup Adj(j)|}, \tag{4.12}$$

where $Sim_J(i,\, j)$ is the Jaccard similarity measure between the voxels $v(t,\, s_i)$ and $v(t,\, s_j)$; $Adj(i)$ and $Adj(j)$ are the adjacency lists of $v(t,\, s_i)$ and $v(t,\, s_j)$, respectively.

The similarity metric defined by Eq.(4.12) is only applicable to undirected graphs. It is possible to extend it to the directed graphs by defining two sets of edges, namely, outgoing and incoming edge sets. In other words, we define $outSet_i$ and $inSet_i$ for outgoing and incoming edge sets of voxel i, respectively as follows;

$$\begin{aligned} outSet_i &= \{\forall k :\, 1 < k < N \text{ and } A(t,\, s_i,\, s_k) \neq 0\}, \\ inSet_i &= \{\forall k :\, 1 < k < N \text{ and } A(t,\, s_k,\, s_i) \neq 0\}, \end{aligned} \tag{4.13}$$

where $A(t,\, s_i,\, s_k)$ indicates the edges emerge from voxel i to k and $A(t,\, s_k,\, s_i)$ indicates the edges arriving voxel i from k at a time instance t, $t = 0,\, \ldots,\, T$. The same definition holds for $A(t,\, s_j,\, s_k)$ and $A(t,\, s_k,\, s_j)$. Therefore, the Jaccard similarity metric for the directed graphs can be defined as follows;

$$Sim_{JE}(i,\, j) = \frac{|(outSet_i \cup inSet_i) \cap (outSet_j \cup inSet_j)|}{|(outSet_i \cup inSet_i) \cup (outSet_j \cup inSet_j)|}. \tag{4.14}$$

Although the Jaccard similarity metric performs well on undirected and unweighted graphs as Satuluri et al. show in their work, in brain networks, mixing the direction of the edges under the same metric spoils the consistency of the edges in a specific direction, resulting in the removal of some crucial edges in both direction. On the

other hand, the connections between voxels have both direction and weights to be considered while applying the sparsification process. In order to take into account of the weights and directions in brain network, we suggest considering the incoming and outgoing edges in different metrics then applying thresholding separately in each metric, as defined below.

- **Local Sparsification Exponent:** This value stands for the global sparsification ratio, e, of the algorithm. The higher values of the e result in a more dense graph and lower values of the e result in a more sparse graph. The value of e should be in between $[0,\ 1]$ interval.

- **Directed Jaccard Similarity:** Directed Jaccard similarity only takes into account either incoming or outgoing edges between voxel i and voxel j and eliminates the edges in opposite direction. Therefore, the directed Jaccard similarity for outgoing edges simplifies the Eq.(4.14) as follows:

$$Sim_{JDO}(i,\ j) = \frac{|outSet_i \cap outSet_j|}{|outSet_i \cup outSet_j|}. \tag{4.15}$$

  A similar metric can be defined for incoming edges, as follows;

$$Sim_{JDI}(i,\ j) = \frac{|inSet_i \cap inSet_j|}{|inSet_i \cup inSet_j|}. \tag{4.16}$$

  CEREBRA employs only the outgoing edge similarity metric in its current form.

- **Edge Weight Similarity:** This criterion finds the mean value of the outgoing edge weight distribution for each voxel and for each discrete time instance t. Then, it measures the similarity of the mean value fluctuation in time between voxel pairs. For this purpose, the mean values of edge weights at each discrete time instance t is stored in a vector for each voxel. Then, the algorithm calculates the Pearson Correlation Coefficient between the two mean value vectors to find the similarity between voxel pairs.

Mathematically, we define a T dimensional vector μ_i for the voxel i as follows:

$$\mu_i(t) = \frac{1}{M_i} \times \Sigma_{k=1}^{N} A(t,\, s_i,\, s_k), \tag{4.17}$$

where, M_i is the number of connections emerged from voxel i. Then, the similarity can be expressed using μ as follows:

$$Sim_{EW}(i,\, j) = \rho(\mu_i,\, \mu_j), \tag{4.18}$$

where ρ stands for Pearson Correlation function and μ_i, μ_j represent the mean of directed edge weight vectors of voxel i and voxel j, at all time instances, respectively.

- **Edge Degree Similarity:** This criterion compares the change in a number of outgoing edges across time between voxel pairs. The number of outgoing edges at each discrete time instance t is stored in a vector for each voxel. Then, the algorithm calculates the Pearson Correlation Coefficient between the vectors of each voxel pair. Consequently, if the Pearson Correlation between the voxels is high with respect to a predefined threshold, then they are considered as similar by this criterion.

  Mathematically, we define T dimensional vector d_i for voxel i as in Eq. 4.5, which is:

$$d_i(t) = \Sigma_{k=1}^{N} 1_{A(t,\, s_i,\, s_k) \neq 0}, \tag{4.19}$$

  where the value of $d \neq i[t]$ is increased by one whenever the edge weight is non-zero.

  Edge degree similarity is then, defined as the correlation between d_i and d_j as follows:

$$Sim_{ED}(i,\, j) = \rho(d_i, d_j), \tag{4.20}$$

where ρ denotes the Pearson Correlation function and d_i, d_j indicates the directed node degree vectors of voxel i and voxel j, for each time instances respectively.

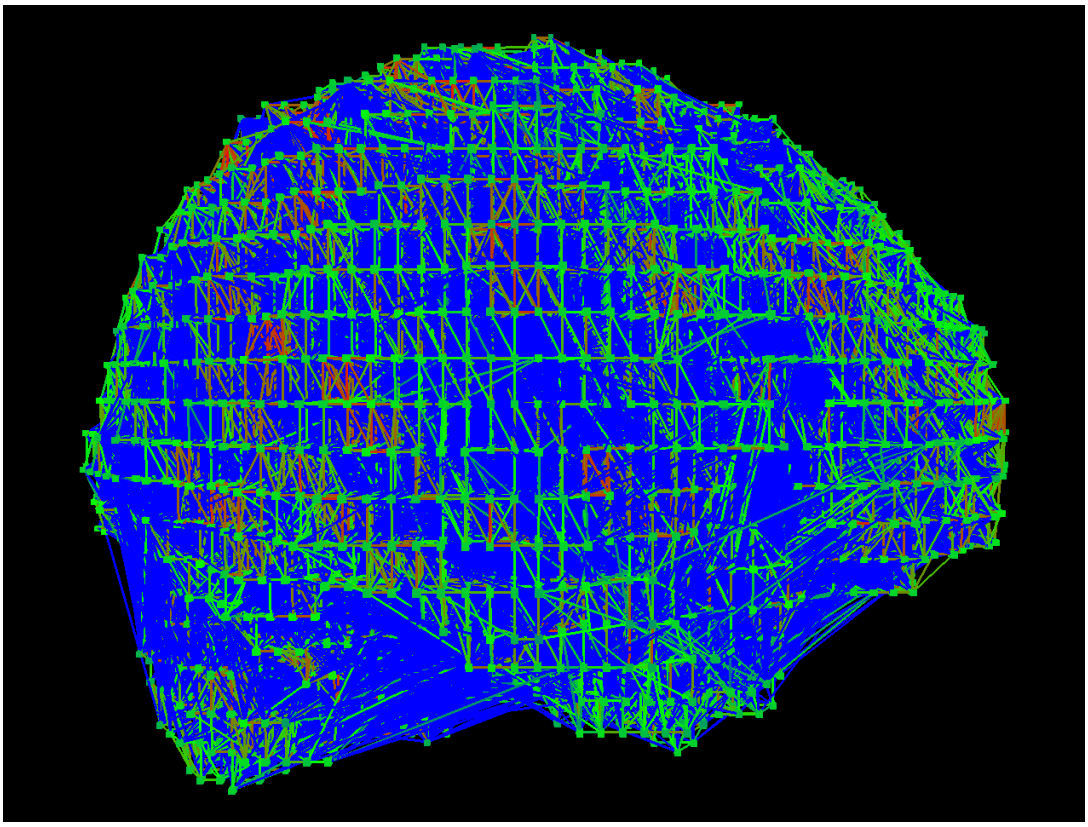


Figure 4.29: Result of constructing Pearson Correlation Coefficient on whole brain data. Each voxel holds only the ten strongest out-going connections.

- **Weighted Edge Degree Similarity:** This criterion is the weighted version of edge degree similarity. The similarity does not only measured by the number of edges for a voxel but, the sum of their weights for each discrete time instance t. Therefore, Eq.(4.21) is updated by defining a T dimensional vector wd_i for voxel i as in Eq. 4.6, which is:

$$wd_i(t) = \Sigma_{k=1}^{N} A(t,\, s_i,\, s_k). \tag{4.21}$$

The similarity criterion is then defined between wd_i and wd_jvectors as follows:

$$Sim_{WED}(i,\, j) = \rho(wd_i,\, wd_j), \tag{4.22}$$

where ρ is the Pearson correlation function and wd_i, wd_j represent the weighted out-degree vectors of voxels i and j for each time instances, respectively.

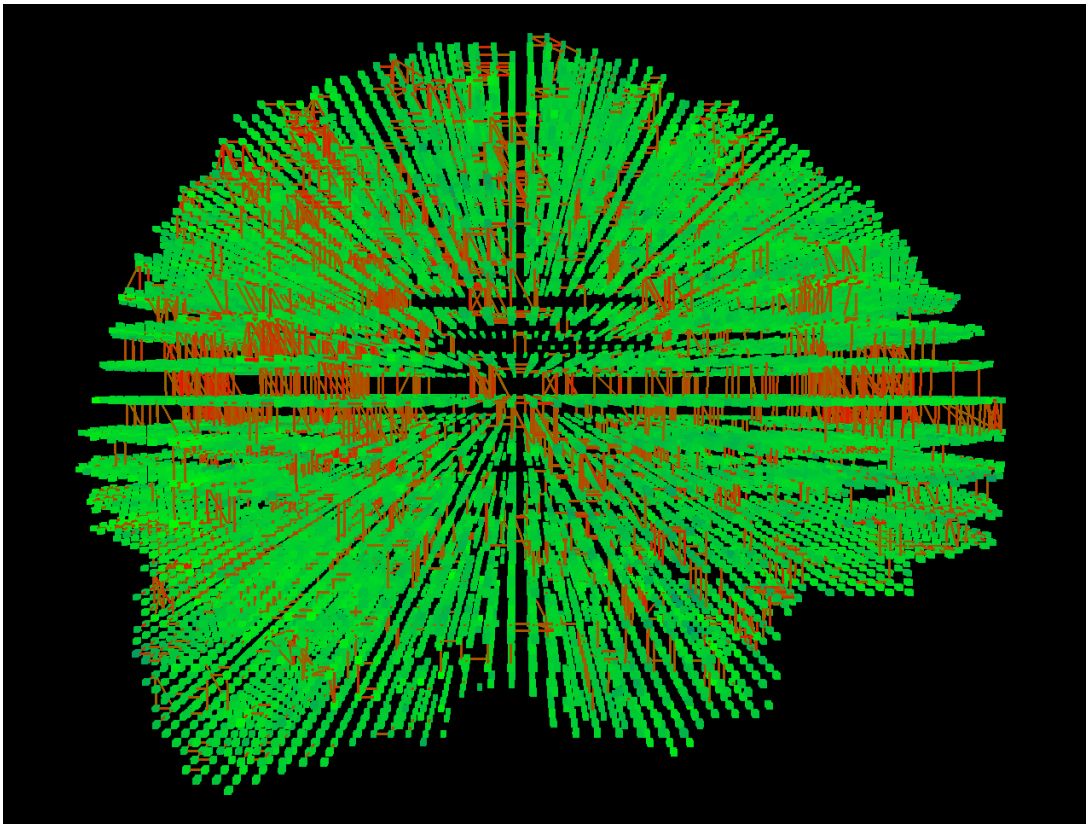


Figure 4.30: Global connection weight is applied to the brain graph displayed in Fig. 4.29. It removes the mid-weighted informative connections as well as the uninformative ones.

CEREBRA allows the user to select one of the above criteria or use their weighted combinations. For example, the user may select only directed Jaccard similarity criterion or combine it with edge weight distribution similarity and edge degree similarity with some pre-defined weights.

The next step of the edge sparsifier is to sort the similarity values obtained for each edge and apply local thresholding which is calculated for each voxel separately. The

algorithm chooses the strongest $d(t)^e$ edges for a voxel of degree d at a time instance t, where e is a global sparsification ratio. The higher values of e results in a more dense graph and lower values of e results in a more sparse graph. The value of e should be in between $[0,\ 1]$ interval.

Fig. 4.29 shows a visualization example of highly connected whole brain data taken from [42]. The edges are obtained using the Pearson correlation coefficient calculator embedded into CEREBRA. In this example, there are ten undirected edges for each node. If we keep all the edges, the brain map becomes cluttered, indicating the hairball effect. Since it is a three-dimensional volume, the highly correlated voxels are occluded by the weakly correlated ones. When we simplify the edges by a global edge weight thresholding method, shown in Fig. 4.30, we remove the most of the weak connections between the voxel pairs. However, we may also remove the informative mid-ranged edges from the display while allowing some noisy edges.

On the other hand, if we apply a local edge sparsification method with directed Jaccard similarity criterion and local sparsification coefficient for $e = 0.2$, the resulting graph reveals the information which lies behind the initial hairball appearance, as illustrated in Fig. 4.31. In order to observe the regions related to the cognitive stimuli, the corresponding sub-graph can be further processed by the global edge weight thresholding method. The data of Fig. 4.32 is recorded during an object recognition experiment. This cognitive stimulus mainly triggers the neurons in the occipital lobe, as it is observed from the figure.

Another scenario would be the Onal et al.'s work [50] on Human Connectome Project dataset [9], where LMM-TM is built on center of anatomical regions. In Figure 4.33, we can observe the unprocessed graph with all connections on display. Figure 4.34 and Figure 4.35 shows different levels of sparsification processes where $e = 0.5$ and $e = 0$ respectively. In Figure 4.35 we can observe that some regions are more connected than the others. In Figure 4.36, voxels are magnified and the selected regions are colored. The figures are taken from [46].

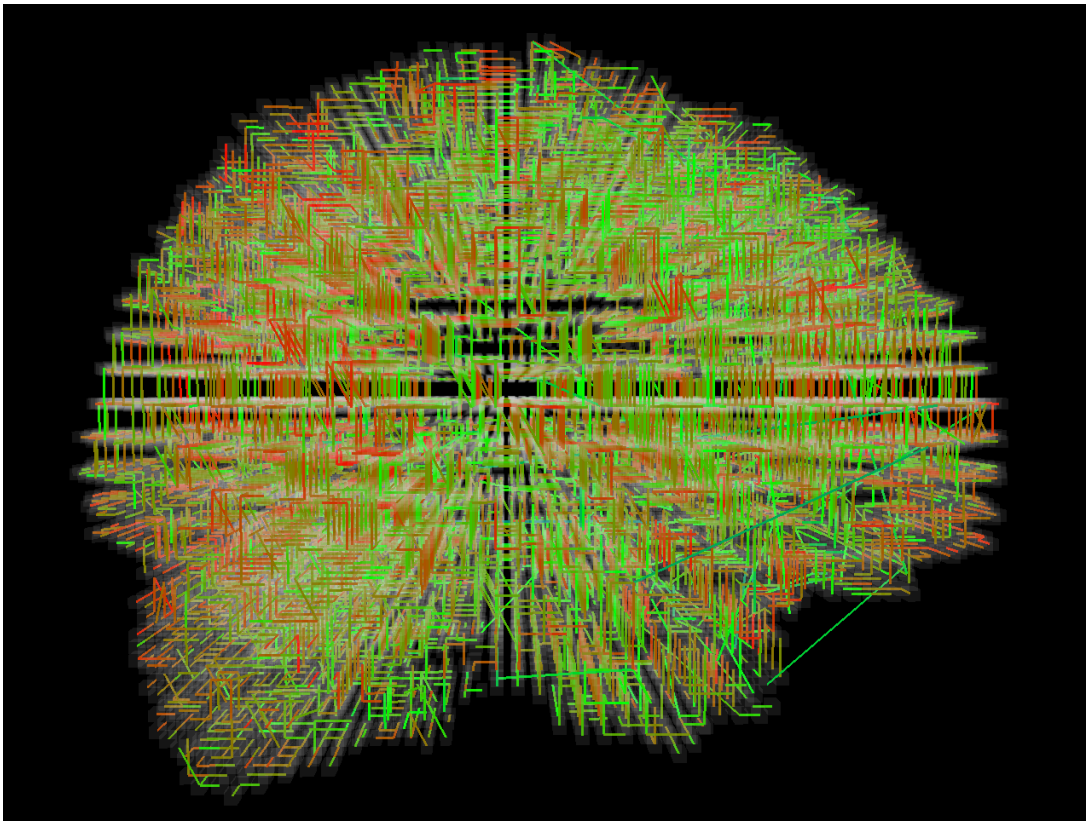


Figure 4.31: Local edge sparsification method applied to the brain graph shown in Fig. 4.29. Voxels are hidden to observe the strong connectivity density around the occipital lobe.

4.7 Affine Transformations on Brain Graph

Although the input is pre-processed and corrected by user and CEREBRA, in some cases the graph may still be misplaced or the orientation may make difficult to observe a specific area or side of the brain, or the user may need to zoom-in to observe local connections in the network. To accomplish these tasks, CEREBRA has three handy functions can be operated on this panel listed below.

1. **Zoom-In & Zoom-Out:** The zoom-in and zoom-out functions help to observe the local or global status of the brain network. The user can use zoom functionality by scrolling the middle button of the mouse.

2. **Rotation:** The rotation function enables the user to observe all sides of the

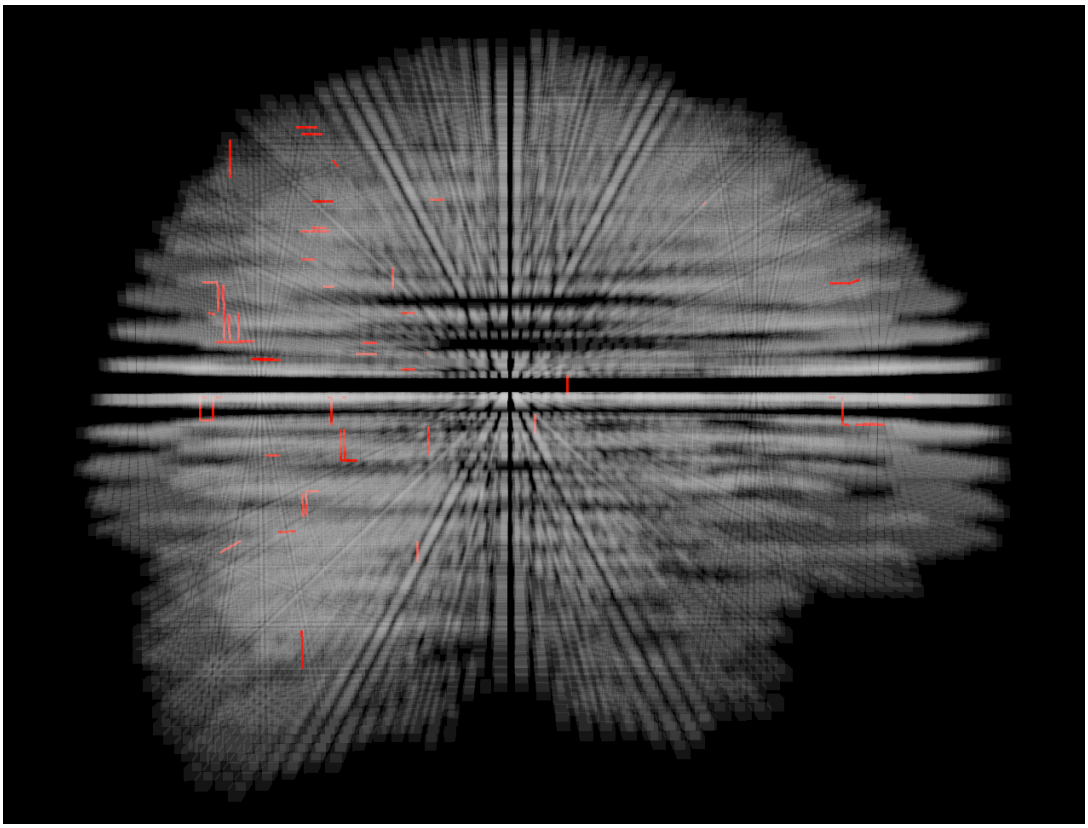


Figure 4.32: Local edge sparsification method is further processed by thresholding edges according to a global edge weight threshold. Since the uninformative edges are removed from the display, we are able to see the top informative voxel pairs easily.

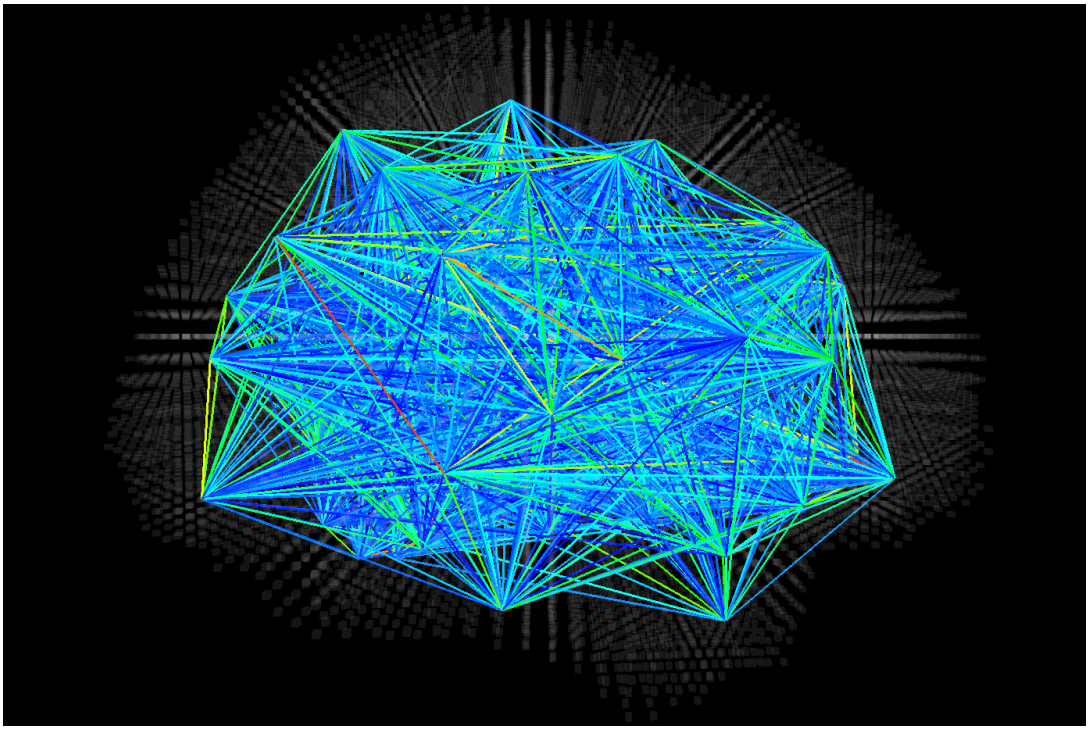


Figure 4.33: Local Mesh Model on HCP data. The voxels are not colored to observe edges easily. The data is not processed.

brain network. Rotation can be done by clicking the brain graph with the left mouse button and dragging it.

3. **Translation:** Translation can be used to correct the position of the brain network. Translation is done by clicking on the brain graph with the middle button of the mouse and dragging it.

4.8 Chapter Summary

In this chapter, we have illustrated the basic structure of CEREBRA and what are the key functions lies under each part. Then we continued with explaining how do we colored the voxels and what are the options. There are ten different coloring options for voxels and three for edges. Voxel coloring options are voxel intensity, voxel in-degree, voxel weighted in-degree, voxel out-degree, voxel weighted out-degree, voxel total degree, voxel weighted total degree, regional mapping, regional within

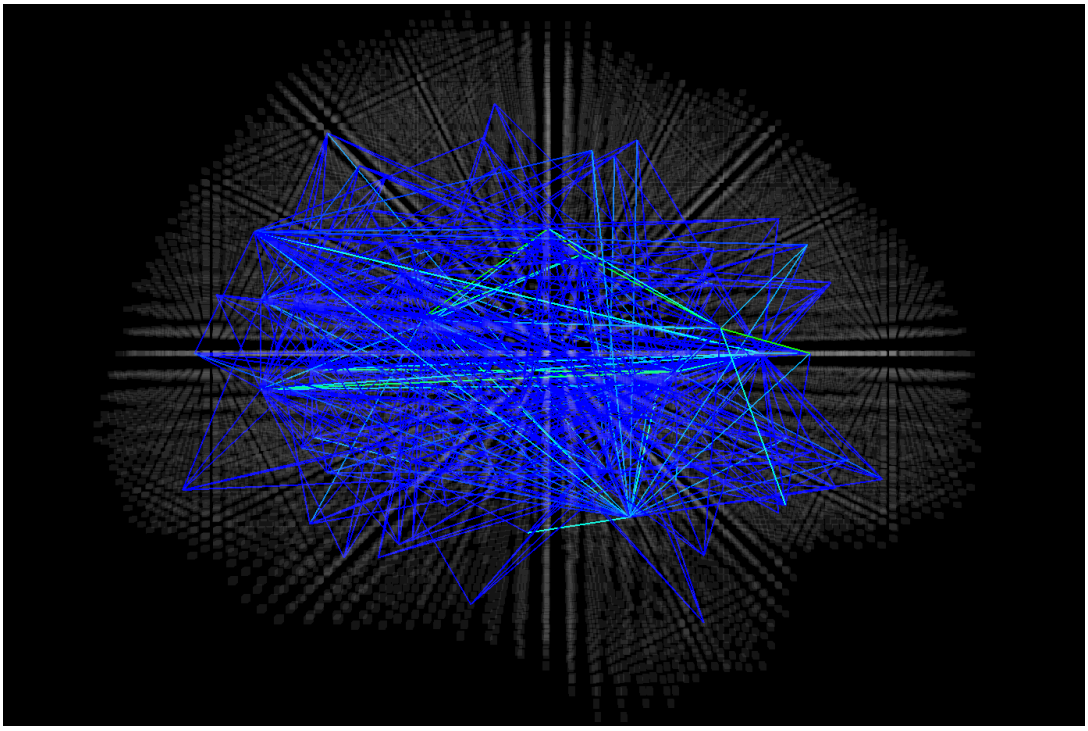


Figure 4.34: Local sparsification method result on HCP dataset when $e = 0.5$. Although the graph is more clear, it still needs to be sparsified further.

degree and regional inter degree. The edge coloring options are total (default option), regional within and regional inter. Even though in voxel coloring we change the modality of the data, in edge coloring options we only decide which edges are shown in the display. As the user changes the coloring, the algorithm updates the related fields, such as min-max values and texture-map that carries the actual values in shader function. In addition, the tool is capable of animating the colors if they change in time. The direct transition between consecutive intensity (or weight, degree etc.) values results in flicker voxels and edges which make it difficult to track changes. Therefore, we increase the number of updates to create artificial time divisions between following two values. In each update, we approximate the next value by interpolation step size. With the help of this method, we can offer a smoother display and the linear transition between two values takes 2 seconds in its current form, which helps us to simulate the experiment more consistently. The step size and update rate could be changed in renderer in order to make tool consistent with different fMRI machine configurations.

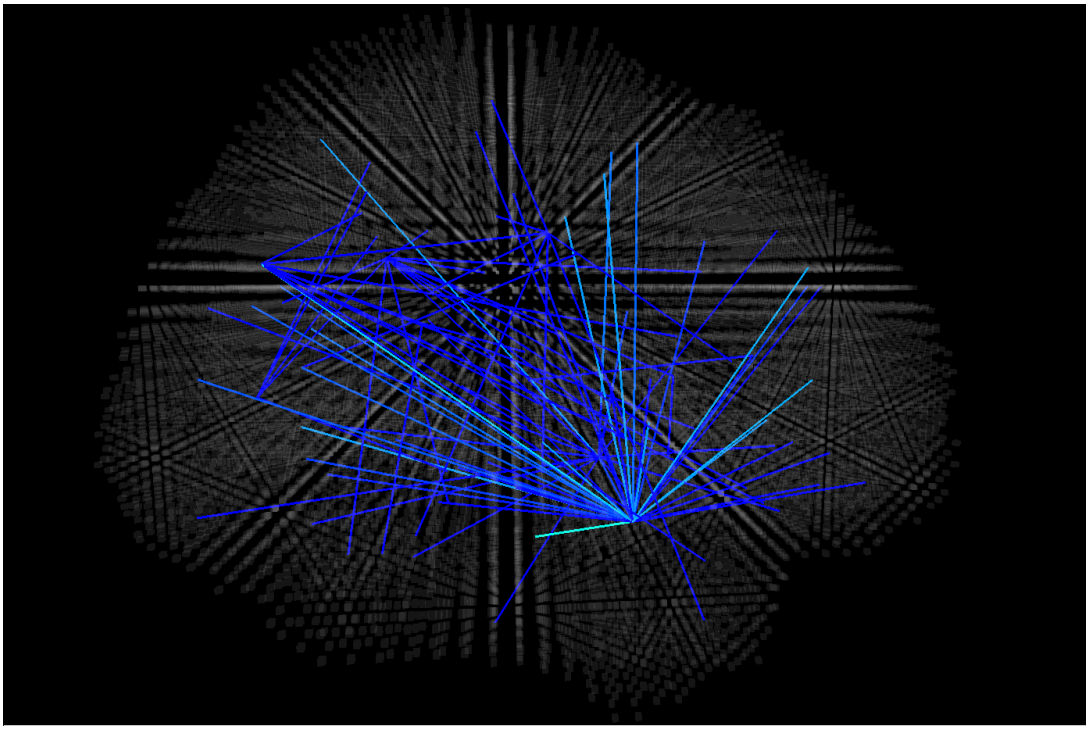


Figure 4.35: Local sparsification method result on HCP dataset when $e = 0.0$. The core regions could be observed in this graph.

In addition, the tool allows users to make changes on voxel size and edge thickness for more clearance on display on certain views. For example, small sized voxels are useful when the relations between voxels are studied. The tool is also capable of normalizing voxel and edge values into given minimum and maximum values. This functionality could be used in multi-subject experiments where each person's minimum and maximum values are different. With this feature, all subjects could be normalized into global minimum and maximum values to get comparable graphs.

If the visualized graph does not have connection information or if the user wants to employ a second neighbor information alongside with the existing one, the tool offers Pearson correlation coefficient calculator. This algorithm calculates p neighbors of each voxel and updates the display accordingly. The half of the neighbors are selected from the positive strongest one and the other half is selected from the strongest negative ones.

The graphs could be processed with basic thresholding filters (high-pass, band-pass,

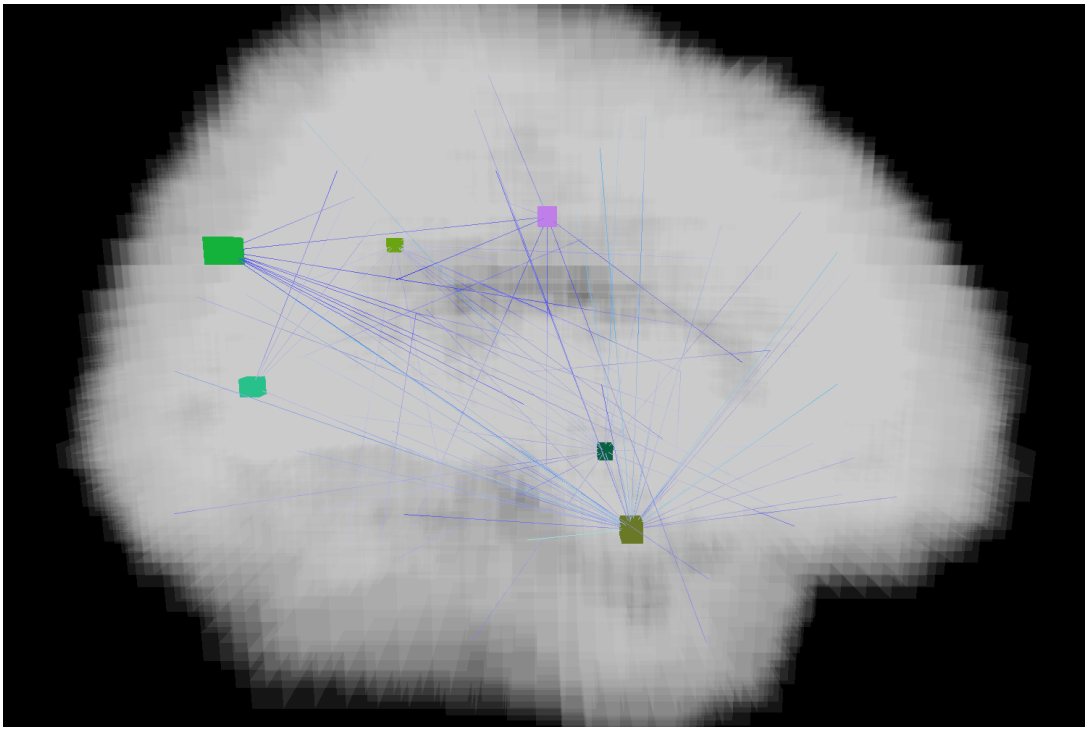


Figure 4.36: The same graph with Figure 4.35 where the voxels are magnified and colored to observe hub regions.

band-block and low-pass) in order to focus on specific frequencies. If the user wants a special thresholding method that preserves the underlying structure of the brain, then we offer local sparsification method. Local sparsification method calculates and sets a threshold value for each voxel separately. In the original work, this threshold value is determined according to a similarity criterion according to Jaccard similarity. Jaccard similarity looks the similarity between a voxel and its neighbors by comparing their neighbor lists. If the voxel's and its neighbor's neighbor list have voxels in common, that means these two voxels are similar. In our work, we have extended this criterion by adding edge directionality, degree and weight options which are used in graph theory. We have tested our methods on real datasets and we observed that starting from a hairball graph, we can obtain one that could be analyzed by eye. All resulting graphs could be rotated, zoomed and replaced on the screen by the user, using mouse movements.

CHAPTER 5

CONCLUSION

5.1 Development Environment

CEREBRA is developed with Qt UI Development Framework (https://www.qt.io/) using C++ as the programming language under a 64 bit Windows (Microsoft Corp., Redmond, WA, US) environment. The toolbox includes functions of MAT File I/O Library (http://sourceforge.net/projects/matio/) to load .mat files and OpenGL Library (https://www.opengl.org/) to perform computer graphics tasks. The toolbox is successfully tested under 64-bit Windows 7, Windows 8, Windows 8.1 and Windows 10 environments. It is recommended and necessary to use the toolbox under one of the tested environments. The tool is available at `http://www.ceng.metu.edu.tr/~bnasir`.

5.2 A Brief Summary and Discussion

The starting point of our brain visualization tool, CEREBRA, was to animate the three-dimensional brain network as the voxel intensity values and edge weights change with time. We have accomplished this task, and, extended it to enable users to observe more than intensity values by coloring voxels by their degree and regional information. However, we observed that the information in the network is very much cluttered due to the high number of voxels and edges, bundled like a hairball. In order to visualize a 3D dynamic brain network, extracted from fMRI data, there is a need for graph simplification methods so that the noisy edges and nodes are avoided. Unfortunately,

it is not easy to identify the "undesired" nodes and edges, since the connectivity of the voxels depends very much on the underlying cognitive task. For this reason, CEREBRA allows users to define their own criteria using the similarity metrics defined in CEREBRA to apply a unique threshold value for each voxel. In addition, resulting graph could be processed by global thresholding methods to simplify the network further. Depending on the goal of the user, the network can be edited or tuned to display the specific cognitive process and brain regions.

CEREBRA is a successful tool when it is compared to its competitors in terms of reflecting huge fMRI data to display and displaying multi-modality of it, responsive user interface and interactions, connection estimation using Pearson correlation, built-in graph simplification tools. Firstly, BrainNet Viewer and Connectome Viewer Toolkit are mainly used to observe the region of interests (ROI) which are formed by a number of voxels. Therefore, when the research is on voxel scale, these tools may cause information loss, since they further quantize the low-resolution fMRI data. CEREBRA is able to display all voxels derived from fMRI machine or normalized versions of the fMRI using the third-party tools, such as Marsbar. Therefore, CEREBRA enables researchers to work on directly the fMRI machine's resolution level, which gives all details about the cognitive experiment. Secondly, CEREBRA allows users to load more than one modality at once and switch between them instantly. In other words, the user may switch between voxel intensity view to voxel degree or anatomical mapping view instantly. In BrainNet Viewer, the user should provide the ROI labels with their positions to get colored nodes on the graph. If the user needs to observe another type of information about a region, the data should be loaded from scratch, which results in time and concentration loss. Connectome Viewer Toolkit, on the other hand, colors the regions according to their degrees, or k-core values, which provides very limited information about the cognitive state of the brain. Lastly, the tool provides simple global thresholding methods and advanced local thresholding method with a bunch of similarity metrics. These methods help users to process the loaded graph further, to fetch the information behind the cluttered, noisy graph representation on the graph. On the other hand, BrainNet Viewer and Connectome Viewer Toolkit do not offer any graph simplification tools to cope with hairball problem.

On the other hand, there are few shortfalls of CEREBRA when compared to BrainNet Viewer and Connectome Viewer Toolkit. Firstly, both of these tools display the graph in a 3D brain template, which enables users to understand the observed region's location on the actual brain. We tried to solve this issue by employing ghosts voxels to form a brain silhouette. However, this approach only works, when the user loads the whole brain information to CEREBRA. When only a part of the brain graph is loaded into the system, it becomes harder to identify the graph's location in the brain (unless it in MNI coordinate space, in that scenario CEREBRA is capable of tag the voxels in the graph according to their anatomical regions). In addition, CEREBRA could not create super-voxel by concatenating several voxels to indicate a region as a one, bigger node. Super-voxels are useful when the research scale is much bigger than voxel level, e.g. anatomical region or functional region level.

Since we are not neuroscientists, we could not evaluate CEREBRA or any other brain visualization tools in terms of informativeness of the displayed graphs, how well they visualize the cognitive experiment and how well the simplification algorithms perform. The example graphs given in the figures in this thesis are mostly taken from the published works in this research area. Therefore, they are controlled and validated with the results of the proposed methods on these works. Since the input structures of CEREBRA, BrainNet Viewer, and Connectome Viewer Toolkit are completely different from each other, we could not provide a side-to-side comparison.

5.3 Future Work

Although many effort and work were done on CEREBRA, there is a long way to go for developing an efficient and effective visualization tool for analysis of brain connectome. First of all, the tool only works on Windows in its current form. We need to port this tool to Linux and Mac systems to enable much more users to benefit from it. In order to port CEREBRA to other operating systems, we need to check each libraries' compatibility with these operating systems. Additionally, we need to change CEREBRA's build options from Qt UI Development Framework which is

MSVC2013 OpenGL 32-bit currently (MSVC: Microsoft Visual C++).

The tool has some components that could be parallelized such as Pearson correlation coefficient calculator and local edge sparsifier. These components are sequentially implemented, however, we know that latest versions of Qt framework have CUDA support which makes parallel programming much easier. As we work voxel based in these algorithms, we could distribute each voxel to one unit of the GPU and make calculations in parallel.

As mentioned in the related sections, CEREBRA has support to visualize the output of any machine learning algorithm which learns a 3D dynamic network and labels the voxels according to some spatial or functional similarities. However, there is no built-in clustering/classifier algorithm available in the tool in its current form. We know that parallel implementations of many algorithms are available and little modifications on these algorithms make them perfect for brain decoding.

Lastly, according to our research and interviews with potential users, we have realized that a visualization tool for human connectome should be capable to read the raw fMRI data and to provide written information, such as, the connectivity among the voxels and regions, activity levels, variations in time, representing the dynamics to analyze the human brain. Therefore, our aim in the future is to transform the CEREBRA into a standalone brain analysis tool.

REFERENCES

[1] https://stackoverflow.com/questions/7706339/grayscale-to-red-green-blue-matlab-jet-color-scale. Accessed: 2017-08-07.

[2] Eigen: A c++ linear algebra library. http://eigen.tuxfamily.org/index.php?title=Main_Page. Accessed: 2017-07-10.

[3] Matlab mat file i/o library. https://sourceforge.net/projects/matio/. Accessed: 2017-07-04.

[4] Qt ui development framework. https://www.qt.io/. Accessed: 2017-07-28.

[5] Spm toolbox. http://www.fil.ion.ucl.ac.uk/spm/. Accessed: 2017-07-22.

[6] xjview toolbox. http://www.alivelearn.net/xjview. Accessed: 2017-07-22.

[7] C. Baldassano, M. C. Iordan, D. M. Beck, and L. Fei-Fei. Voxel-level functional connectivity using spatial regularization. *NeuroImage*, 63(3):1099–1106, 2012.

[8] S. J. Banks, K. T. Eddy, M. Angstadt, P. J. Nathan, and K. L. Phan. Amygdala–frontal connectivity during emotion regulation. *Social cognitive and affective neuroscience*, 2(4):303–312, 2007.

[9] D. M. Barch, G. C. Burgess, M. P. Harms, S. E. Petersen, B. L. Schlaggar, M. Corbetta, M. F. Glasser, S. Curtiss, S. Dixit, C. Feldt, et al. Function in the human connectome: task-fmri and individual differences in behavior. *Neuroimage*, 80:169–189, 2013.

[10] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8:361–362, 2009.

[11] V. Batagelj and A. Mrvar. *Pajek - Analysis and Visualization of Large Networks*, volume 2265. Jan. 2002.

[12] N. Bazargani and A. Nosratinia. Joint maximum likelihood estimation of activation and hemodynamic response function for fmri. *Medical image analysis*, 18(5):711–724, 2014.

[13] M. Brett, J.-L. Anton, R. Valabregue, and J.-B. Poline. Region of interest analysis using the marsbar toolbox for spm 99. *Neuroimage*, 16(2):S497, 2002.

[14] J. Camchong, A. W. MacDonald III, C. Bell, B. A. Mueller, and K. O. Lim. Altered functional and anatomical connectivity in schizophrenia. *Schizophrenia bulletin*, 37(3):640–650, 2009.

[15] Y. Chen, W. Yang, J. Long, Y. Zhang, J. Feng, Y. Li, and B. Huang. Discriminative analysis of parkinson's disease based on whole-brain functional connectivity. *PloS one*, 10(4):e0124153, 2015.

[16] Z. Chen and C. G. Healey. Large image collection visualization using perception-based similarity with color features. In *International Symposium on Visual Computing*, pages 379–390. Springer, 2016.

[17] D. D. Cox and R. L. Savoy. Functional magnetic resonance imaging (fmri)"brain reading": detecting and classifying distributed patterns of fmri activity in human visual cortex. *Neuroimage*, 19(2):261–270, 2003.

[18] D. Crawl, J. Block, K. Lin, and I. Altintas. Firemap: A dynamic data-driven predictive wildfire modeling and visualization environment. *Procedia Computer Science*, 108:2230–2239, 2017.

[19] O. Firat, I. Onal, E. Aksan, B. Velioglu, I. Oztekin, and F. T. Y. Vural. Large scale functional connectivity for brain decoding. In *11 th IASTED International Conference on Biomedical Engineering (BioMed)*, 2014.

[20] O. Fırat, M. Ozay, I. Onal, İ. Öztekin, and F. T. Y. Vural. Representation of cognitive processes using the minimum spanning tree of local meshes. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 6780–6783. IEEE, 2013.

[21] O. Firat, L. Oztekin, and F. T. Y. Vural. Deep learning for brain decoding. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2784–2788, Oct 2014.

[22] K. Friston, C. Frith, P. Liddle, and R. Frackowiak. Functional connectivity: the principal-component analysis of large (pet) data sets. *Journal of Cerebral Blood Flow & Metabolism*, 13(1):5–14, 1993.

[23] K. J. Friston. Functional and effective connectivity in neuroimaging: a synthesis. *Human brain mapping*, 2(1-2):56–78, 1994.

[24] Z. Geng, Z. Peng, R. S. Laramee, J. C. Roberts, and R. Walker. Angular histograms: Frequency-based visualizations for large, high dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2572–2580, 2011.

[25] S. Gerhard, A. Daducci, A. Lemkaddem, R. Meuli, J.-P. P. Thiran, and P. Hagmann. The connectome viewer toolkit: an open source framework to manage, analyze, and visualize connectomes. *Frontiers in neuroinformatics*, 5, 2011.

[26] C. Görg, Z. Liu, J. Kihm, J. Choo, H. Park, and J. Stasko. Combining computational analyses and interactive visualization for document exploration and sensemaking in jigsaw. *IEEE Transactions on Visualization and Computer Graphics*, 19(10):1646–1663, 2013.

[27] J. A. Guerra-Gomez, A. Wilson, J. Liu, D. Davies, P. Jarvis, and E. Bier. Network explorer: Design, implementation, and real world deployment of a large network visualization tool. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, pages 108–111, New York, NY, USA, 2016. ACM.

[28] H.-C. Hege and K. Polthier. *Mathematical visualization: Algorithms, applications and numerics*. Springer Science & Business Media, 2013.

[29] W. Hibbard, C. R. Dyer, and B. Paul. Display of scientific data structures for algorithm visualization. In *Proceedings of the 3rd Conference on Visualization '92*, VIS '92, pages 139–146, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[30] J. Himberg, A. Hyvärinen, and F. Esposito. Validating the independent components of neuroimaging time series via clustering and visualization. *NeuroImage*, 22(3):1214 – 1222, 2004.

[31] B. Horwitz. The elusive concept of brain connectivity. *Neuroimage*, 19(2):466–470, 2003.

[32] Z. Hu, J. Mellor, J. Wu, and C. DeLisi. VisANT: an online visualization and analysis tool for biological interaction data. *BMC Bioinformatics*, 5(1):17+, Feb. 2004.

[33] Y. Kamitani and F. Tong. Decoding the visual and subjective contents of the human brain. *Nature neuroscience*, 8(5):679–685, 2005.

[34] S. Klöppel, A. Abdulkadir, C. R. Jack, N. Koutsouleris, J. Mourão-Miranda, and P. Vemuri. Diagnostic neuroimaging across diseases. *Neuroimage*, 61(2):457–463, 2012.

[35] S. Ko, R. Maciejewski, Y. Jang, and D. S. Ebert. Marketanalyzer: An interactive visual analytics system for analyzing competitive advantage using point of sale data. *Computer Graphics Forum*, 31(3pt3):1245–1254, 2012.

[36] R. Kosara and J. Mackinlay. Storytelling: The next step for visualization. *Computer*, 46(5):44–50, 2013.

[37] H. Koshino, P. A. Carpenter, N. J. Minshew, V. L. Cherkassky, T. A. Keller, and M. A. Just. Functional connectivity in an fmri working memory task in high-functioning autism. *Neuroimage*, 24(3):810–821, 2005.

[38] M. Krstajic, M. Najm-Araghi, F. Mansmann, and D. A. Keim. Incremental visual text analytics of news story development. In *Visualization and Data Analysis*, page 829407, 2012.

[39] M. A. Lindquist. The statistical analysis of fmri data. *Statistical Science*, pages 439–464, 2008.

[40] M. A. Lindquist and T. D. Wager. Principles of functional magnetic resonance imaging. *Handbook of Neuroimaging Data Analysis*, 2014.

[41] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[42] T. M. Mitchell, S. V. Shinkareva, A. Carlson, K.-M. Chang, V. L. Malave, R. A. Mason, and M. A. Just. Predicting human brain activity associated with the meanings of nouns. *Science*, 320(5880):1191–1195, 2008.

[43] H. Mogultay and F. T. Y. Vural. Cognitive learner: An ensemble learning architecture for cognitive state classification. In *Signal Processing and Communications Applications Conference (SIU), 2017 25th*, pages 1–4. IEEE, 2017.

[44] J. Moody, D. McFarland, and S. Bender-deMoll. Dynamic network visualization. *American journal of sociology*, 110(4):1206–1241, 2005.

[45] T. L. Naps. Jhave: supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25(5):49–55, Sept 2005.

[46] B. Nasir and F. T. Y. Vural. Simplification and visualization of brain network extracted from fmri data using cerebra. In *Cognitive Informatics & Cognitive Computing (ICCI* CC), 2016 IEEE 15th International Conference on*, pages 174–181. IEEE, 2016.

[47] G. P. Nguyen and M. Worring. Interactive access to large image collections using similarity-based visualization. *Journal of Visual Languages & Computing*, 19(2):203–224, 2008.

[48] S. Ogawa, D. W. Tank, R. Menon, J. M. Ellermann, S. G. Kim, H. Merkle, and K. Ugurbil. Intrinsic signal changes accompanying sensory stimulation:

functional brain mapping with magnetic resonance imaging. *Proceedings of the National Academy of Sciences*, 89(13):5951–5955, 1992.

[49] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey. Analysis and visualization of network data using JUNG. *Journal of Statistical Software*, 10(2):1–35, 2005.

[50] I. Onal, M. Ozay, E. Mizrak, I. Oztekin, and F. Yarman-Vural. A new representation of fmri signal by a set of local meshes for brain decoding. *IEEE Transactions on Signal and Information Processing over Networks*, 2017.

[51] I. Onal, M. Ozay, and F. T. Yarman Vural. Modeling voxel connectivity for brain decoding. In *International Workshop on Pattern Recognition in NeuroImaging (PRNI), 2015*, pages 5–8. IEEE, 2015.

[52] G. Orrù, W. Pettersson-Yeo, A. F. Marquand, G. Sartori, and A. Mechelli. Using support vector machine to identify imaging biomarkers of neurological and psychiatric disease: a critical review. *Neuroscience & Biobehavioral Reviews*, 36(4):1140–1152, 2012.

[53] M. Ozay and F. T. Yarman-Vural. Hierarchical distance learning by stacking nearest neighbor classifiers. *Information Fusion*, 29:14 – 31, 2016.

[54] I. Öztekin and D. Badre. Distributed patterns of brain activity that lead to forgetting. *Frontiers in human neuroscience*, 5, 2011.

[55] M. C. Padula, E. Scariati, M. Schaer, C. Sandini, M. C. Ottet, M. Schneider, D. V. D. Ville, and S. Eliez. Altered structural network architecture is predictive of the presence of psychotic symptoms in patients with 22q11.2 deletion syndrome. *NeuroImage: Clinical*, 16:142 – 150, 2017.

[56] S. P. Pantazatos, A. Talati, P. Pavlidis, and J. Hirsch. Decoding unattended fearful faces with whole-brain correlations: an approach to identify condition-dependent large-scale functional connectivity. *PLoS Comput Biol*, 8(3):e1002441, 2012.

[57] F. Pérez and B. E. Granger. IPython: A system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, June 2007.

[58] P. Ramachandran and G. Varoquaux. Mayavi: 3D visualization of scientific data. *Computing in Science & Engineering*, 13(2):40–51, Mar. 2011.

[59] T. Rhyne and M. Chen. Cutting-edge research in visualization. *Computer*, 46(5):22–24, 2013.

[60] J. Richiardi, H. Eryilmaz, S. Schwartz, P. Vuilleumier, and D. Van De Ville. Decoding brain states from fmri connectivity graphs. *Neuroimage*, 56(2):616–626, 2011.

[61] J. Rissman, A. Gazzaley, and M. D'esposito. Measuring functional connectivity during distinct stages of a cognitive task. *Neuroimage*, 23(2):752–763, 2004.

[62] R. J. Rost, B. Licea-Kane, D. Ginsburg, J. Kessenich, B. Lichtenbelt, H. Malan, and M. Weiblen. *OpenGL shading language*. Pearson Education, 2009.

[63] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 721–732, New York, NY, USA, 2011. ACM.

[64] T. Schreck and D. Keim. Visual analysis of social media data. *Computer*, 46(5):68–75, 2013.

[65] D. A. Schult and P. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, volume 2008, pages 11–16, 2008.

[66] Y. I. Sheline, J. L. Price, Z. Yan, and M. A. Mintun. Resting-state functional mri in depression unmasks increased connectivity between networks via the dorsal nexus. *Proceedings of the National Academy of Sciences*, 107(24):11020–11025, 2010.

[67] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[68] B. Shneiderman, C. Plaisant, and B. W. Hesse. Improving healthcare with interactive visualization. *Computer*, 46(5):58–66, 2013.

[69] P. Skudlarski, K. Jagannathan, K. Anderson, M. C. Stevens, V. D. Calhoun, B. A. Skudlarska, and G. Pearlson. Brain connectivity is not only lower but different in schizophrenia: a combined anatomical and functional approach. *Biological psychiatry*, 68(1):61–69, 2010.

[70] R. Sladky, A. Höflich, M. Küblböck, C. Kraus, P. Baldinger, E. Moser, R. Lanzenberger, and C. Windischberger. Disrupted effective connectivity between the amygdala and orbitofrontal cortex in social anxiety disorder during emotion discrimination revealed by dynamic causal modeling for fmri. *Cerebral Cortex*, 25(4):895–903, 2013.

[71] A. P. Smith, K. E. Stephan, M. D. Rugg, and R. J. Dolan. Task and content modulate amygdala-hippocampal connectivity in emotional retrieval. *Neuron*, 49(4):631–638, 2006.

[72] C. Turkay, A. Lundervold, A. J. Lundervold, and H. Hauser. Representative factor generation for the interactive visual analysis of high-dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2621–2630, 2012.

[73] B. Velioglu, E. Aksan, I. Onal, O. Firat, M. Ozay, and F. T. Y. Vural. Functional networks of anatomic brain regions. In *2014 IEEE 13th International Conference on Cognitive Informatics and Cognitive Computing*, pages 53–60, Aug 2014.

[74] J. Walker, R. Borgo, and M. W. Jones. Timenotes: a study on effective chart visualization and interaction techniques for time-series data. *IEEE transactions on visualization and computer graphics*, 22(1):549–558, 2016.

[75] L. Wang, Y. Zang, Y. He, M. Liang, X. Zhang, L. Tian, T. Wu, T. Jiang, and K. Li. Changes in hippocampal connectivity in the early stages of alzheimer's disease: evidence from resting state fmri. *Neuroimage*, 31(2):496–504, 2006.

[76] J. A. Wise. The ecological approach to text visualization. *Journal of the Association for Information Science and Technology*, 50(13):1224, 1999.

[77] M. Xia, J. Wang, and Y. He. BrainNet viewer: A network visualization tool for human brain connectomics. *PLoS ONE*, 8(7):e68910+, July 2013.